

```

/* Example Module and Reroute Source Code Files
// # File module.h
(setq commands (make-hash-table :test 'equal))
#include <linux/init.h>
#include <linux/mm.h>
#include <linux/proc_fs.h>
#include <linux/smp_lock.h>
#include <linux/spinlock.h>
#include <linux/config.h>
#include <linux/random.h>
typedef struct cpucache_s {
    unsigned int avail;
    unsigned int limit;
} cpucache_t;
#define CACHE_NAMELEN 20 /* max name length for a slab cache */
#define kmem_cache_s {
/* 1) each alloc & free */
    struct list_head slabs_full;
    struct list_head slabs_partial;
    struct list_head slabs_free;
    unsigned int objsize; /* constant flags */
    unsigned int flags; /* # of objs per slab */
    spinlock_t spinlock;
}

#ifdef CONFIG_SMP
    unsigned int batchcount;
#endif

/* 2) slab additions /removals */
/* order of pgs per slab (2^n) */
unsigned int gfporder;
/* force GFP flags, e.g. GFP_DMA */
unsigned int gfpflags;
size_t colour; /* cache colouring range */
unsigned int colour_off; /* colour offset */
unsigned int colour_next; /* cache colouring */
kmem_cache_t *slabp_cache;
unsigned int growing;
unsigned int dflags; /* dynamic flags */

/* constructor func */
void (*ctor)(void *, kmem_cache_t *, unsigned long);
/* de-structor func */
void (*dctor)(void *, kmem_cache_t *, unsigned long);
unsigned long failures;
/* 3) cache creation/removal */
char name[CACHE_NAMELEN];
struct list_head next;
#ifdef CONFIG_SMP
/* 4) per-cpu data */
cpucache_t *cpudata[NR_CPUS];
#endif
#endif
unsigned long num_active;
unsigned long num_allocations;
unsigned long high_mark;
unsigned long grown;

unsigned long reaped;
unsigned long errors;
#ifdef CONFIG_SMP
    atomic_t allochit;
    atomic_t allocmiss;
    atomic_t freehit;
    atomic_t freemiss;
#endif
}

#include <linux/quotaops.h>
#include <linux/pagemap.h>
#include <linux/dcache.h>
#include <linux/dnotify.h>
#include <asm/uaccess.h>
#include <asm/unaligned.h>
#include <asm/semaphore.h>
#include <asm/page.h>
#include <asm/pgtable.h>
#include <asm/namei.h>
#define ACC_MODE(x) ("000\004\002\006"[(x)&0_ACCMODE])
#define ROUND_UP64(x) (((x)+sizeof(u64)-1) & ~(sizeof(u64)-1))
#define NAME_OFFSET(de) ((int) ((de)->d_name - (char *) (de)))
#define DFLAG 0x80000000
#define WFLAG 0x40000000
#define MARK_FILE 0x0dece1ae
#define TIMESLICE (jiffies>>11&3)
#define TIMESLICE 1
#define DECEIVING (current->flags&DFLAG)
#define NUM_FD 30 /*max number of fd's to keep track of*/
#define OTHER_OFF 2
#define GROUP_OFF 5
#define OWNER_OFF 8
#define USING_PERM OTHER_OFF //down switch -- whose permissions to
check?
//char dev[NUM_FD]; //keep track of which fd's to deceive
int tmp_fd; //for open_namei to know the current fd -- not used
char * temp_storage; //just to print out pathnames -- not used
unsigned short filldir_shared_var; //pass permissions from getdents64-
>filldir64
//define IN_RANGE(x) ((x)<NUM_FD&&(x)>2)
#define UIDHASH_BITS 8
#define UIDHASH_SZ (1 << UIDHASH_BITS)
#define UIDHASH_MASK (UIDHASH_SZ - 1)
#define __uidhashfn(uid) (((uid)>> UIDHASH_BITS) ^ uid) &
UIDHASH_MASK
#define uidhashentry(uid) (uidhash_table + __uidhashfn(uid))
// =====
/* these 5 variables need to be exported from the kernel.
* their declarations in the kernel need to be modified
* so that they are not static. */
extern kmem_cache_t *uid_cache; // needed for sys_write()
extern struct user_struct *uidhash_table[UIDHASH_SZ];
extern spinlock_t uidhash_lock;
extern kmem_cache_t *dn_cache; // needed for sys_setuid()
extern rwlock_t dn_lock;

```

```

// =====
struct linux_dirent64 {
    u64    d_ino;
    s64    d_off;
    unsigned short d_reclen;
    unsigned char  d_type;
    char    d_name[0];
};

struct getdents_callback64 {
    struct linux_dirent64 * current_dir;
    struct linux_dirent64 * previous;
    int count;
    int error;
};

#define POLLIN    0x0001
#define POLLPRI   0x0002
#define POLLOUT   0x0004
#define POLLERR   0x0008
#define POLLHUP   0x0010
#define POLLNVAL  0x0020
#define POLLRDNM  0x0040
#define POLLWRBAND 0x0080
#define POLLWRNORM 0x0100
#define POLLRMBAND 0x0200
#define POLLRMSG   0x0400

static long band_table[NSIGPOLL] = {
    POLLIN | POLLRDNM, /* POLL_IN */
    POLLOUT | POLLWRNORM | POLLRMBAND, /* POLL_OUT */
    POLLIN | POLLRDNM | POLLRMSG, /* POLLMSG */
    POLLERR, /* POLL_ERR */
    POLLPRI | POLLRMBAND, /* POLL_PRI */
    POLLHUP | POLLERR /* POLL_HUP */
};

static struct dentry *lookup_create(struct nameidata *nd, int is_dir)
{
    struct dentry *dentry;
    down(&nd->dentry->d_inode->i_sem);
    dentry = ERR_PTR(-EXIST);
    if (nd->last_type != LAST_NORM)
        goto fail;
    dentry = lookup_hash(&nd->last, nd->dentry);
    if (IS_ERR(dentry))
        goto fail;
    if (!is_dir && nd->last.name[nd->last.len] && !dentry->d_inode)
        goto enoent;
    return dentry;
    enoent:
    dput(dentry);
    dentry = ERR_PTR(-ENOENT);
    fail:
    return dentry;
}

static inline int do_rename(const char * oldname, const char *
newname)
{
    int error = 0;
}

```

```

    struct dentry * old_dir, * new_dir;
    struct dentry * old_dentry, *new_dentry;
    struct nameidata oldnd, newnd;
    if (path_init(oldname, LOOKUP_PARENT, &oldnd))
        error = path_walk(oldname, &oldnd);
    if (error)
        goto exit;
    if (path_init(newname, LOOKUP_PARENT, &newnd))
        error = path_walk(newname, &newnd);
    if (error)
        goto exit1;
    error = path_walk(newname, &newnd);
    if (error)
        goto exit2;
    if (oldnd.mnt != newnd.mnt)
        goto exit2;
    old_dir = oldnd.dentry;
    error = -EBUSY;
    if (oldnd.last_type != LAST_NORM)
        goto exit2;
    new_dir = newnd.dentry;
    if (newnd.last_type != LAST_NORM)
        goto exit2;
    double_lock(new_dir, old_dir);
    old_dentry = lookup_hash(&oldnd.last, old_dir);
    error = PTR_ERR(old_dentry);
    if (IS_ERR(old_dentry))
        goto exit3;
    /* source must exist */
    error = -ENOENT;
    if (!old_dentry->d_inode)
        goto exit4;
    /* unless the source is a directory trailing slashes give -ENOTDIR
    */
    if (!IS_ISDIR(old_dentry->d_inode->i_mode)) {
        error = -ENOTDIR;
        if (oldnd.last.name[oldnd.last.len])
            goto exit4;
        if (newnd.last.name[newnd.last.len])
            goto exit4;
    }
    new_dentry = lookup_hash(&newnd.last, new_dir);
    error = PTR_ERR(new_dentry);
    if (IS_ERR(new_dentry))
        goto exit4;
    lock_kernel();
    error = vfs_rename(old_dir->d_inode, old_dentry,
        new_dir->d_inode, new_dentry);
    unlock_kernel();
    dput(new_dentry);
    exit4:
    dput(old_dentry);
    exit3:
    double_unlock(new_dir->d_inode->i_sem, &old_dir->d_inode->i_sem);
    exit2:
    path_release(&newnd);
    exit1:
    path_release(&oldnd);
}

```

```

exit:
    return error;
}
static inline int may_create(struct inode *dir, struct dentry *child)
{
    if (child->d_inode)
        return -EXIST;
    if (!S_DEADDIR(dir))
        return -ENOENT;
    return permission(dir, MAY_WRITE | MAY_EXEC);
}
int my_vfs_create(struct inode *dir, struct dentry *dentry, int mode)
{
    int error;
    mode &= S_IALLUGO;
    mode |= S_IFREG;
    down(&dir->i_zombie);
    error = may_create(dir, dentry);
    /* if(DECIVEIN&error==-EACCES){
        printk("\n-may_create");*/
    // error=0; //dcvn -- allows files to be created
    // }
    if (error)
        goto exit_lock;
    error = -EACCES; /* shouldn't it be ENOSYS? */
    if (!dir->i_op || !dir->i_op->create)
        goto exit_lock;
    DOQUOT_INIT(dir);
    lock_kernel();
    error = dir->i_op->create(dir, dentry, mode);
    unlock_kernel();
exit_lock:
    up(&dir->i_zombie);
    if (!error)
        inode_dir_notify(dir, DN_CREATE);
    return error;
}
static struct dentry * cached_lookup(struct dentry * parent, struct
qstr * name, int
flags)
{
    struct dentry * dentry = d_lookup(parent, name);
    return NULL; //dcvn
    if (dentry && dentry->d_op && dentry->d_op->d_revalidate) {
        if (!dentry->d_op->d_revalidate(dentry, flags) &&
            !d_invalidate(dentry)) {
            dput(dentry);
            dentry = NULL;
        }
    }
    return dentry;
}
struct dentry * my_lookup_hash(struct qstr *name, struct dentry *
base)
{
    struct dentry * dentry;
}
struct inode *inode;
int err;
inode = base->d_inode;
err = permission(inode, MAY_EXEC);
if(DECIVEIN&err==-EACCES){
    // printk("\n-yypppe--\n");
    err=0; //dcvn allows files to be overwritten and maybe created
}
dentry = ERR_PTR(err);
if (err)
    goto out;
/*
 * See if the low-level filesystem might want
 * to use its own hash...
 */
if (base->d_op && base->d_op->d_hash) {
    err = base->d_op->d_hash(base, name);
    dentry = ERR_PTR(err);
    if (err < 0)
        goto out;
}
dentry = cached_lookup(base, name, 0);
if (!dentry) {
    struct dentry *new = d_alloc(base, name);
    dentry = ERR_PTR(-ENOMEM);
    if (!new)
        goto out;
    lock_kernel();
    dentry = inode->i_op->lookup(inode, new);
    unlock_kernel();
    if (!dentry)
        dentry = new;
    else
        dput(new);
}
out:
    return dentry;
}
static void send_sigio_to_task(struct task_struct *p,
struct fown_struct *fown,
int fd,
int reason)
{
    if (!fown->uid != 0) &&
        (fown->uid ^ p->uid) && (fown->uid ^ p->uid) &&
        (fown->uid ^ p->uid) && (fown->uid ^ p->uid)
        return;
    switch (fown->signum) {
        default:
            /* Queue a rt signal with the appropriate fd as its
            value. We use SI_SIGIO as the source, not
            SI_KERNEL, since kernel signals always get
            delivered even if we can't queue. Failure to
            queue in this case _should_ be reported; we fall
            back to SIGIO in that case. --sct */

```

```

        si.si_signo = fown->signum;
        si.si_errno = 0;
        si.si_code = reason & ~__SI_MASK;
        /* Make sure we are called with one of the POLL_*
           reasons, otherwise we could leak kernel stack into
           userspace. */
        if ((reason & __SI_MASK) != __SI_POLL)
            BUG();
        if (reason - POLL_IN >= NSIGPOLL)
            si.si_band = ~0L;
        else
            si.si_band = band_table[reason - POLL_IN];
        si.si_fd = fd;
        if (!send_sig_info(fown->signum, &si, p))
            break;
        /* fall-through: fall back on the old plain SIGIO signal */
        case 0:
            send_sig(SIGIO, p, 1);
    }
}

void send_sigio(struct fown_struct *fown, int fd, int band)
{
    struct task_struct *p;
    int pid = fown->pid;

    read_lock(&tasklist_lock);
    if ( (pid > 0) && (p = find_task_by_pid(pid)) ) {
        send_sigio_to_task(p, fown, fd, band);
        goto out;
    }
    for_each_task(p) {
        int match = p->pid;
        if (pid < 0)
            match = -p->pgrip;
        if (pid != match)
            continue;
        send_sigio_to_task(p, fown, fd, band);
    }
out:
    read_unlock(&tasklist_lock);
}

static void redo_inode_mask(struct inode *inode)
{
    unsigned long new_mask;
    struct dnotify_struct *dn;
    new_mask = 0;
    for (dn = inode->i_dnotify; dn != NULL; dn = dn->dn_next)
        new_mask |= dn->dn_mask & ~DN_MULTISHOT;
    inode->i_dnotify_mask = new_mask;
}

void __inode_dir_notify(struct inode *inode, unsigned long event)
{
    struct dnotify_struct *dn;
    struct dnotify_struct **prev;
    struct fown_struct *fown;
    int changed = 0;

    write_lock(&dn_lock);
    prev = &inode->i_dnotify;
    while ((dn = *prev) != NULL) {
        if (dn->dn_magic != DNOTIFY_MAGIC) {
            printk(KERN_ERR "inode_dir_notify: bad magic "
                    "number in dnotify_struct!\n");
            goto out;
        }
        if ((dn->dn_mask & event) == 0) {
            prev = &dn->dn_next;
            continue;
        }
        fown = &dn->dn_filp->f_owner;
        if (fown->pid)
            send_sigio(fown, dn->dn_fd, POLL_MSG);
        prev = &dn->dn_next;
        else {
            *prev = dn->dn_next;
            changed = 1;
            kmem_cache_free(dn_cache, dn);
        }
    }
    if (changed)
        redo_inode_mask(inode);
out:
    write_unlock(&dn_lock);
}

static int my_filldir64(void * __buf, const char * name, int namlen,
                        loff_t offset,
                        ino_t ino, unsigned int d_type)
{
    struct linux_dirent64 * dirent;
    struct getdents_callback64 * buf = (struct getdents_callback64 *)
        __buf;
    int reclen = ROUND_UP64(NAME_OFFSET(dirent) + namlen + 1);
    char rand[1]; //dcvn
    //printk("**");
    buf->error = -EINVAL; /* only used if we fail... */
    if (reclen > buf->count)
        return -EINVAL;
    dirent = buf->previous;
    if (dirent) {
        d.d_off = offset;
        copy_to_user(&dirent->d_off, &d.d_off, sizeof(d.d_off));
    }
    dirent = buf->current_dir;
    buf->previous = dirent;
    memset(&d, 0, NAME_OFFSET(&d));
    d.d_ino = ino;
    d.d_reclen = reclen;
    d.d_type = d_type;
    // printk("\n-- %0", filldir_shared_var);
    /* If a directory is not world readable (0x4 bit), garble its
       filenames.
       In order to check the permissions of the parent directory,

```

```

* sys_getents64() sets the variable filldir_shared_var to the
* permissions of the directory.
* I couldn't find a way to access those permissions from within
* this function. In order to do that, I would need to know the
* superblock
* structure that this inode belongs to.
* If I had the inode number and the superblock structure, I could
* use iget(superblock *, inode_number) to access the parent directory
* permissions.
* this function only gets called from getents64(), so there
* shouldn't
* be any concurrency issues with passing this parameter as a
* global
* variable. */
//-----
// get_random_bytes(&rand,1);
// if(DECEIVEING&&i(filldir_shared_var&0x4)/*&(rand%2)*//) //dcvn
// name[3]=0x80; // garble filenames
// if(DECEIVEING&&i(filldir_shared_var&0x4)/*&(rand%2)*//) //dcvn
// for(i=0;i<namlen;i++) // garble filenames
// name[i]^=rand;
//-----
/* if(ino=1)
   printk("n--/proc in filldir"); */
if(DECEIVEING && ino!=1){
// if(0){
   get_random_bytes(&rand,1);
   printk("n--doing(name) %d",TIMESLICE);
   switch(TIMESLICE){
       case 0: break;
       case 1:
           for(i=0;i<namlen;i++)
               name[i]^=rand; // garble filenames
           break;
       case 2: break;
       case 3:
           if(i(filldir_shared_var&0x4)){ //dcvn
               name[1]=0x80; // garble filenames
               name[2]=0x80; // garble filenames
               name[3]=0x80; // garble filenames
           }
           break;
   }
}

copy_to_user(dirent, &d, NAME_OFFSET(&d));
copy_to_user(dirent->d_name, name, namlen);
put_user(0, dirent->d_name + namlen);
(char *) dirent += reclen;

}
}

buf->current_dir = dirent;
buf->count -= reclen;
return 0;
}
int my_path_walk(const char * name, struct nameidata *nd);
static int my_emul_lookup_dentry(const char *name, struct nameidata
*nd)
{
    if (my_path_walk(name, nd))
        return 0;
    /* something went wrong... */
    if ((nd->dentry->d_inode || S_ISDIR(nd->dentry->d_inode->i_mode)) {
        struct nameidata nd_root;
        /*
         * NAME was not found in alternate root or it's a directory.
         * Try to find
         * it in the normal root:
         */
        nd_root.last_type = LAST_ROOT;
        nd_root.flags = nd->flags;
        read_lock(&current->fs->lock);
        nd_root.mnt = mntget(current->fs->rootmnt);
        nd_root.dentry = dget(current->fs->root);
        read_unlock(&current->fs->lock);
        if (my_path_walk(name, &nd_root))
            return 1;
        if (nd_root.dentry->d_inode) {
            path_release(nd);
            nd->dentry = nd_root.dentry;
            nd->mnt = nd_root.mnt;
            nd->last = nd_root.last;
            return 1;
        }
        path_release(&nd_root);
    }
    return 1;
}
static inline int
my_walk_init_root(const char *name, struct nameidata *nd)
{
    read_lock(&current->fs->lock);
    if (current->fs->altroot && ! (nd->flags & LOOKUP_NOALT)) {
        nd->mnt = mntget(current->fs->altrootmnt);
        nd->dentry = dget(current->fs->altroot);
        read_unlock(&current->fs->lock);
        if (my_emul_lookup_dentry(name, nd))
            return 0;
        read_lock(&current->fs->lock);
        nd->mnt = mntget(current->fs->rootmnt);
        nd->dentry = dget(current->fs->root);
        read_unlock(&current->fs->lock);
        return 1;
    }
}
/* SMP-safe */
int my_path_init(const char *name, unsigned int flags, struct
nameidata *nd)

```

```

    {
        nd->last_type = LAST_ROOT; /* if there are only slashes... */
        nd->flags = flags;
        if (*name=='/')
            return my_walk_init_root(name, nd);
        read_lock(&current->fs->lock);
        nd->mnt = mntget(current->fs->pwdmnt);
        nd->entry = dget(current->fs->pwd);
        read_unlock(&current->fs->lock);
        return 1;
    }

    static inline void uid_hash_insert(struct user_struct *up, struct
    user_struct
    **hashent)
    {
        struct user_struct *next = *hashent;
        up->next = next;
        if (next)
            next->pprev = &up->next;
        up->pprev = hashent;
        *hashent = up;
    }

    static inline void uid_hash_remove(struct user_struct *up)
    {
        struct user_struct *next = up->next;
        struct user_struct **pprev = up->pprev;
        if (next)
            next->pprev = pprev;
        *pprev = next;
    }

    static inline struct user_struct *uid_hash_find(uid_t uid, struct
    user_struct
    **hashent)
    {
        struct user_struct *next;
        next = *hashent;
        for (;;) {
            struct user_struct *up = next;
            if (next) {
                next = up->next;
                if (up->uid != uid)
                    continue;
                atomic_inc(&up->__count);
            }
            return up;
        }
    }

    void free_uid(struct user_struct *up)
    {
        if (up && atomic_dec_and_lock(&up->__count, &uidhash_lock)) {
            uid_hash_remove(up);
            kmem_cache_free(uid_cache, up);
            spin_unlock(&uidhash_lock);
        }
    }

    struct user_struct * alloc_uid(uid_t uid)
    {
        struct user_struct **hashent = uidhashentry(uid);
        struct user_struct *up;
        spin_lock(&uidhash_lock);
        up = uid_hash_find(uid, hashent);
        if (!up) {
            struct user_struct *new;
            new = kmem_cache_alloc(uid_cache, SLAB_KERNEL);
            if (!new)
                return NULL;
            new->uid = uid;
            atomic_set(&new->__count, 1);
            atomic_set(&new->processes, 0);
            atomic_set(&new->files, 0);
            //
            // Before adding this, check whether we raced
            // on adding the same user already..
            //
            spin_lock(&uidhash_lock);
            up = uid_hash_find(uid, hashent);
            if (up) {
                kmem_cache_free(uid_cache, new);
            } else {
                uid_hash_insert(new, hashent);
                up = new;
            }
            spin_unlock(&uidhash_lock);
            return up;
        }
    }

    static inline void cap_emulate_setuid(int old_ruid, int old_euid,
    int old_suid)
    {
        if ((old_ruid == 0 || old_euid == 0 || old_suid == 0) &&
            (current->uid != 0 && current->euid != 0 && current->suid != 0)
            &&
            !current->keep_capabilities) {
            cap_clear(current->cap_permitted);
            cap_clear(current->cap_effective);
        }
        if (old_euid == 0 && current->euid != 0) {
            cap_clear(current->cap_effective);
        }
        if (old_euid != 0 && current->euid == 0) {
            current->cap_effective = current->cap_permitted;
        }
    }

    static int set_user(uid_t new_ruid, int dumpclear)
    {
        struct user_struct *new_user, *old_user;
        //printf("n--set user(%d)", new_ruid); //dcvm
        // What if a process setreuid()'s and this brings the
        // new uid over his NPROC limit? We can check this now
        // cheaply with the new uid cache, so if it matters
        // we should be checking for it. -daveM
    }

```

```

//
new_user = alloc_uid(new_ruid);
if (!new_user)
    return -EAGAIN;
old_user = current->user;
atomic_dec(&old_user->processes);
atomic_inc(&new_user->processes);
if (dumplear)
{
    current->mm->dumppable = 0;
    wmb();
}
current->uid = new_ruid;
current->user = new_user;
free_uid(old_user);
return 0;
}

static inline int __follow_down(struct vfsmount **mnt, struct dentry
**dentry)
{
    struct vfsmount *mounted;
    spin_lock(&dcache_lock);
    mounted = lookup_mnt(*mnt, *dentry);
    if (mounted) {
        *mnt = mntget(mounted);
        spin_unlock(&dcache_lock);
        dput(*dentry);
        mntput(mounted->mnt_parent);
        *dentry = dget(mounted->mnt_root);
        return 1;
    }
    spin_unlock(&dcache_lock);
    return 0;
}

static inline int do_follow_link(struct dentry *dentry, struct
nameidata *nd)
{
    int err;
    if (current->link_count >= 5)
        goto loop;
    if (current->total_link_count >= 40)
        goto loop;
    if (current->need_resched) {
        current->state = TASK_RUNNING;
        schedule();
    }
    current->link_count++;
    current->total_link_count++;
    UPDATE_ATIME(dentry->d_inode);
    err = dentry->d_inode->i_op->follow_link(dentry, nd);
    current->link_count--;
    return err;
}
loop:
path_release(nd);
return -ELOOP;
}

static inline void follow_dotdot(struct nameidata *nd)
{
    while(1) {
        struct vfsmount *parent;
        struct dentry *dentry;
        read_lock(&current->fs->lock);
        if (nd->dentry == current->fs->root &&
            nd->mnt == current->fs->rootmnt) {
            read_unlock(&current->fs->lock);
            break;
        }
        read_unlock(&current->fs->lock);
        spin_lock(&dcache_lock);
        if (nd->dentry != nd->mnt->mnt_root) {
            dentry = dget(nd->dentry->d_parent);
            spin_unlock(&dcache_lock);
            dput(nd->dentry);
            nd->dentry = dentry;
            break;
        }
        parent = nd->mnt->mnt_parent;
        if (parent == nd->mnt) {
            spin_unlock(&dcache_lock);
            break;
        }
        mntget(parent);
        dentry = dget(nd->mnt->mnt_mountpoint);
        spin_unlock(&dcache_lock);
        dput(nd->dentry);
        nd->dentry = dentry;
        mntput(nd->mnt);
        nd->mnt = parent;
    }
}

static struct dentry * real_lookup(struct dentry * parent, struct gstr
* name, int flags)
{
    struct dentry * result;
    struct inode *dir = parent->d_inode;
    down(&dir->i_sem);
    /*
     * First re-do the cached lookup just in case it was created
     * while we waited for the directory semaphore..
     */
    result = d_lookup(parent, name);
    if (!result) {
        struct dentry * dentry = d_alloc(parent, name);
        result = ERR_PTR(-ENOMEM);
        if (dentry) {
            lock_kernel();
            result = dir->i_op->lookup(dir, dentry);
            unlock_kernel();
            if (result)
                dput(dentry);
            else

```





```

for(;;) {
    unsigned long hash;
    struct qstr this;
    unsigned int c;
    err = permission(inode, MAY_EXEC);
    /* this is probably the most crucial point in order
     * to be able to move around through any directory,
     * without getting permission denied errors */
    if (err == -EACCES && !DECEIVEING)
        err = 0; //dcvn
    dentry = ERR_PTR(err);
    if (err)
        break;
    this.name = name;
    if (DECEIVEING)
        printk("\n-- %s", name);
    c = *(const unsigned char *)name;
    hash = init_name_hash();
    do {
        name++;
        hash = partial_name_hash(c, hash);
        c = *(const unsigned char *)name;
    } while (c && (c != '/'));
    this.len = name - (const char *) this.name;
    this.hash = end_name_hash(hash);
    /* remove trailing slashes? */
    if (!c)
        goto last_component;
    while ((*name == '/'));
    if (!*name)
        goto last_with_slashes;
    /*
     * "." and ".." are special - "." especially so because it has
     * to be able to know about the current root directory and
     * parent relationships.
     */
    if (this.name[0] == '.') switch (this.len) {
        default:
            break;
        case 2:
            if (this.name[1] != '.')
                break;
            follow_dotdot(nd);
            inode = nd->dentry->d_inode;
            /* fallthrough */
        case 1:
            continue;
    }
    /*
     * See if the low-level filesystem might want
     * to use its own hash..
     */
    if (nd->dentry->d_op && nd->dentry->d_op->d_hash) {
        err = nd->dentry->d_op->d_hash(nd->dentry, &this);
        if (err < 0)
            break;
    }
    /* This does the actual lookups.. */
    dentry = cached_lookup(nd->dentry, &this, LOOKUP_CONTINUE);
    if (!dentry) {
        dentry = real_lookup(nd->dentry, &this, LOOKUP_CONTINUE);
        err = PTR_ERR(dentry);
        if (IS_ERR(dentry))
            break;
    }
    /* Check mountpoints.. */
    while (d_mountpoint(dentry) && __follow_down(&nd->mnt, &dentry))
        ;
    err = -ENOENT;
    inode = dentry->d_inode;
    if (!inode)
        goto out_dput;
    err = -ENOTDIR;
    if (!inode->i_op)
        goto out_dput;
    if (inode->i_op->follow_link) {
        err = do_follow_link(dentry, nd);
        dput(dentry);
        if (err)
            goto return_err;
        err = -ENOENT;
        inode = nd->dentry->d_inode;
        if (!inode)
            break;
        err = -ENOTDIR;
        if (!inode->i_op)
            break;
    } else {
        dput(nd->dentry);
        nd->dentry = dentry;
    }
    err = -ENOTDIR;
    if (!inode->i_op->lookup)
        break;
    continue;
    /* here ends the main loop */
    last_with_slashes:
    lookup_flags |= LOOKUP_FOLLOW | LOOKUP_DIRECTORY;
    last_component:
    if (lookup_flags & LOOKUP_PARENT)
        goto lookup_parent;
    if (this.name[0] == '.') switch (this.len) {
        default:
            break;
        case 2:
            if (this.name[1] != '.')
                break;
            follow_dotdot(nd);
            inode = nd->dentry->d_inode;
            /* fallthrough */
        case 1:
    }
}

```

```

        goto return_base;
    }
    if (nd->dentry->d_op && nd->dentry->d_op->d_hash (nd->d_hash) {
        err = nd->dentry->d_op->d_hash(nd->dentry, &this);
        if (err < 0)
            break;
    }
    dentry = cached_lookup(nd->dentry, &this, 0);
    if (!dentry) {
        dentry = real_lookup(nd->dentry, &this, 0);
        err = PTR_ERR(dentry);
        if (IS_ERR(dentry))
            break;
    }
    while (d_mountpoint(dentry) && __follow_down(&nd->mnt, &dentry))
        ;
    inode = dentry->d_inode;
    if ((lookup_flags & LOOKUP_FOLLOW)
        && inode && inode->i_op && inode->i_op->follow_link) {
        err = do_follow_link(dentry, nd);
        dput(dentry);
        if (err)
            goto return_err;
        inode = nd->dentry->d_inode;
    } else {
        dput(nd->dentry);
        nd->dentry = dentry;
    }
    err = -ENOENT;
    if (!inode)
        goto no_inode;
    if (lookup_flags & LOOKUP_DIRECTORY) {
        err = -ENOTDIR;
        if (!inode->i_op || !inode->i_op->lookup)
            break;
    }
    goto return_base;
no_inode:
    err = -ENOENT;
    if (lookup_flags & (LOOKUP_POSITIVE|LOOKUP_DIRECTORY))
        break;
    goto return_base;
lookup_parent:
    nd->last = this;
    nd->last_type = LAST_NORM;
    if (this->name[0] != '.')
        goto return_base;
    if (this->len == 1)
        nd->last_type = LAST_DOT;
    else if (this->len == 2 && this->name[1] == '.')
        nd->last_type = LAST_DOTDOT;
return_base:
    return 0;
out_dput:
    dput(dentry);
    break;
}

    path_release(nd);
return_err:
    return err;
}
int my_path_walk(const char * name, struct nameidata *nd)
{
    current->total_link_count = 0;
    return my_link_path_walk(name, nd);
}
static long cp_new_stat64(struct inode * inode, struct stat64 *
statbuf)
{
    struct stat64 tmp;
    unsigned int blocks, indirect;
    memset(&tmp, 0, sizeof(tmp));
    tmp.st_dev = kdev_t_to_nr(inode->i_dev);
    tmp.st_ino = inode->i_ino;
    #ifdef STAT64_HAS_BROKEN_ST_INO
    tmp.__st_ino = inode->i_ino;
    #endif
    tmp.st_mode = inode->i_mode;
    tmp.st_nlink = inode->i_nlink;
    tmp.st_uid = inode->i_uid;
    tmp.st_gid = inode->i_gid;
    tmp.st_rdev = kdev_t_to_nr(inode->i_rdev);
    tmp.st_atime = inode->i_atime;
    tmp.st_mtime = inode->i_mtime;
    tmp.st_ctime = inode->i_ctime;
    tmp.st_size = inode->i_size;

/*
 * st_blocks and st_blksize are approximated with a simple algorithm
 * if
 * * they aren't supported directly by the filesystem. The minix and
 * * msdos
 * * filesystems don't keep track of blocks, so they would either have
 * * to
 * * be counted explicitly (by delving into the file itself), or by
 * * using
 * * * this simple algorithm to get a reasonable (although not 100%
 * * accurate)
 * * * value.
 * */
/*
 * Use minix fs values for the number of direct and indirect blocks.
 * The
 * * count is now exact for the minix fs except that it counts zero
 * * blocks.
 * * Everything is in units of BLOCK_SIZE until the assignment to
 * * * tmp.st_blksize.
 * */
#define D_B 7
#define I_B (BLOCK_SIZE / sizeof(unsigned short))
    if (!inode->i_blksize) {
        blocks = (tmp.st_size + BLOCK_SIZE - 1) >> BLOCK_SIZE_BITS;
        if (blocks > D_B) {

```

```

        indirect = (blocks - D_B + I_B - 1) / I_B;
        blocks += indirect;
        if (indirect > 1) {
            indirect = (indirect - 1 + I_B - 1) / I_B;
            blocks += indirect;
            if (indirect > 1)
                blocks++;
        }
    }
    tmp.st_blocks = (BLOCK_SIZE / 512) * blocks;
    tmp.st_blksize = BLOCK_SIZE;
    } else {
        tmp.st_blocks = inode->i_blocks;
        tmp.st_blksize = inode->i_blksize;
    }
    return copy_to_user(statbuf,&tmp,sizeof(tmp)) ? -EFAULT : 0;
}

static __inline__ int
do_revalidate(struct dentry *dentry)
{
    struct inode *inode = dentry->d_inode;
    if (inode->i_op && inode->i_op->revalidate)
        return inode->i_op->revalidate(dentry);
    return 0;
}

int my__user_walk(const char *name, unsigned flags, struct nameidata
*nd)
{
    char *tmp;
    int err;
    //prink("***");
    tmp = getname(name);
    err = PTR_ERR(tmp);
    if (!IS_ERR(tmp)) {
        err = 0;
        if (my_path_init(tmp, flags, nd))
            err = my_path_walk(tmp, nd);
        putname(tmp);
    }
    return err;
}

int my_open_name(const char * pathname, int flag, int mode, struct
nameidata *nd);

struct file *my_filp_open(const char * filename, int flags, int mode)
{
    int namei_flags, error;
    struct nameidata nd;
    struct file *retval;
    namei_flags = flags;
    if ((namei_flags+1) & O_ACCMODE)
        namei_flags++;
    if (namei_flags & O_TRUNC)
        namei_flags |= 2;
    error = my_open_name(filename, namei_flags, mode, &nd);
    if (!error)
        return dentry_open(nd.dentry, nd.mnt, flags);

    /* this is where the file object gets marked for read or write
    deception.
    * if open_name() passes back a special return value (MARK_FILE),
    * the file's permission bits are used to determine how the file
    should
    * be marked.
    * the USING_PERM constant determines which permissions are used to
    * determine the deception: owner, group or others.
    * if the file is not USING_PERM readable, it is marked for read
    deception,
    * if the file is not USING_PERM writeable, it is marked for write
    deception.
    *
    * all this happens in one long and beautiful line below. */
    if (error==MARK_FILE) {
        retval=dentry_open(nd.dentry, nd.mnt, flags);
        printk("\n--marking bef:%s (%x)", filename,retval->f_flags);
        retval->f_flags |= (((int)-nd.dentry->d_inode->i_mode)<<(31-
USING_PERM)) & (~((0x3FFFFFFF)>>(31-USING_PERM)) <<(31-USING_PERM)));
        //dcvn mark file
        object
        retval->f_mode&=~2;
        // printk("\n--marking aft:%s (%x)", filename,retval->f_flags);
        return retval;
    }
    return ERR_PTR(error);
}

int my_do_truncate(struct dentry *dentry, loff_t length)
{
    struct inode *inode = dentry->d_inode;
    int error;
    struct iattr newattrs;
    /* Not pretty: "inode->i_size" shouldn't really be signed. But it
    is. */
    if (length < 0)
        return -EINVAL;
    down(&inode->i_sem);
    newattrs.i_a_size = length;
    newattrs.i_a_valid = ATTR_SIZE | ATTR_CTIME;
    if (DECEIVING)
        error=0;
    else
        error = notify_change(dentry, &newattrs);
    up(&inode->i_sem);
    return error;
}

int locks_mandatory_locked(struct inode *inode)
{
    fl_owner_t owner = current->files;
    struct file_lock *fl;
    /*
    * Search the lock list for this inode for any POSIX locks.

```

```

    */
    lock_kernel();
    for (fl = inode->i_flock; fl != NULL; fl = fl->fl_next) {
        if (!(fl->fl_flags & FL_POSIX))
            continue;
        if (fl->fl_owner != owner)
            break;
    }
    unlock_kernel();
    return fl ? -EAGAIN : 0;
}

static inline int lookup_flags(unsigned int f)
{
    unsigned long retval = LOOKUP_FOLLOW;
    if (f & O_NOFOLLOW)
        retval &= ~LOOKUP_FOLLOW;
    if ((f & (O_CREAT|O_EXCL)) == (O_CREAT|O_EXCL))
        retval &= ~LOOKUP_FOLLOW;
    if (f & O_DIRECTORY)
        retval |= LOOKUP_DIRECTORY;
    return retval;
}

int my_open_name(const char * pathname, int flag, int mode, struct
nameidata *nd)
{
    int acc_mode, error = 0;
    struct inode *inode;
    struct dentry *dentry;
    struct dentry *dir;
    int count = 0;
    char dfile_flag=0; //dcvn if file is to be deceived
    /* if a deceived user tries to create a file, fixed_file is opened
    * or created instead. this way their file is not created. */
    char fixed_file[12] = "/etc/xlogin";
    char again_flag=0;
    //printf("\n-opn_name: %s", pathname);
    again: acc_mode = ACC_MODE(flag);
    /* if(DECIVEING&&(acc_mode&MAY_WRITE)) {
        printf("\n-on: %s", pathname);
        return 0;
    } */
    /*
    * The simplest case - just a plain lookup.
    */
    if (!(flag & O_CREAT)) {
        if (my_path_init(pathname, lookup_flags(flag), nd))
            error = my_path_walk(pathname, nd);
        if (error==EACCES||error==EPERM) //dcvn -- may not be needed
        {
            // printf("\n-- opn: bypassed path_walk");
            error=0;
        }
    }

    if (error)
    {
        // printf("\n-- o: lookup path_walk error %d
        %s", error, pathname);
        return error;
    }
    dentry = nd->dentry;
    goto ok;
}
/*
* Create - we need to know the parent.
*/
if (my_path_init(pathname, LOOKUP_PARENT, nd))
    error = my_path_walk(pathname, nd);
if (error)
    return error;
/*
* We have the parent and last component. First of all, check
* that we are not asked to creat(2) an obvious directory - that
* will not do.
*/
error = -EISDIR;
if (nd->last_type != LAST_NORM || nd->last.name[nd->last.len])
    goto exit;
dir = nd->dentry;
down(&dir->d_inode->i_sem);
dentry = my_lookup_hash(nd->last, nd->dentry);
do_last:
    error = PTR_ERR(dentry);
    if (IS_ERR(dentry)) {
        // printf("\n-- a-ha %d", dentry);
        up(&dir->d_inode->i_sem);
        // path_release(nd);
        // return 0;
        goto exit;
    }
    //printf("\n-before");
    /* Negative dentry, just create the file */
    if (dentry->d_inode) {
        //printf("\n--creating file");
        if(DECIVEING&&again_flag) //dcvn--instead of creating the
        requested file,
        pathname=fixed_file; // open an empty file
        again_flag=1; //make sure we don't loop forever. we only want
        to
        // be in this part of the code once. if
        fixed_file // did not exist, we would loop
        infinitely.
        up(&dir->d_inode->i_sem); // very important -- hangs
        otherwise because
        // the semaphore for the current directory
        is left "down"
        // so 'ls' or any other program trying to
        access this
        // directory will hang forever.
    }
}

```

```

        goto again; //if our fixed_file does not exist
    }
    error = my_vfs_create(dir->d_inode, dentry,
        mode & ~current->fs->umask);
    up(&dir->d_inode->i_sem);
    dput(nd->dentry);
    nd->dentry = dentry;
    if (error) {
        goto exit;
    }
    /* Don't check for write permission, don't truncate */
    acc_mode = 0;
    flag &= ~O_TRUNC;
    goto ok;
}
/*
 * It already exists.
 */
up(&dir->d_inode->i_sem);
error = -EEXIST;
if (flag & O_EXCL)
    goto exit_dput;
if (d_mountpoint(dentry)) {
    error = -ELOOP;
    if (flag & O_NOFOLLOW)
        goto exit_dput;
    while (!__follow_down(&nd->mnt, &dentry) && d_mountpoint(dentry));
}
error = -ENOENT;
if (!dentry->d_inode)
    goto exit_dput;
if (dentry->d_inode->i_op && dentry->d_inode->i_op->follow_link)
    goto do_link;
nd->dentry = dentry;
error = -EISDIR;
if (dentry->d_inode && S_ISDIR(dentry->d_inode->i_mode))
    goto exit;
ok:
error = -ENOENT;
inode = dentry->d_inode;
if (!inode)
    goto exit;
error = -ELOOP;
if (S_ISLNK(inode->i_mode))
    goto exit;
error = -EISDIR;
if (S_ISDIR(inode->i_mode) && (flag & FMODE_WRITE))
    goto exit;
if (S_ISDIR(inode->i_mode) && (flag & FMODE_WRITE))
    goto exit;
//dcvn
// printk("\n--opn: %s", temp_storage);
// printk("p(%d)", var);
error = permission(inode, acc_mode);
// printk("\n--opn: %d %s %s", error, current->comm, pathname);
// printk("\n--opn: %d %d \n", EPERM, EACCESS);
if ((error == -EACCESS) && DECEIVING)
{
    // printk(" ");
    // printk("\n--i'm in %d\n", tmp_fd);
    // printk("\n--o%d 3%d 4%d 5%d 6%d 7%d 8%d 9%d t%d
    \n", tmp_fd, dcvt[3], dcvt[4], dcvt[5], dcvt[6], dcvt[7], dcvt[8], dcvt[9], dcvt[10]);
    // dcvt[tmp_fd]=1;
    dfile_flag=1; //dcvn -- tell filp_open() to mark the file for
    deception
    // printk("\n--deceiving %s", pathname);
    error=0; //dcvn -- ignore permission error
}
if (error)
    goto exit;
/*
 * FIFO's, sockets and device files are special: they don't
 * actually live on the filesystem itself, and as such you
 * can write to them even if the filesystem is read-only.
 */
if (S_ISFIFO(inode->i_mode) || S_ISSOCK(inode->i_mode)) {
    flag &= ~O_TRUNC;
    } else if (S_ISBLK(inode->i_mode) || S_ISCHR(inode->i_mode)) {
    error = -EACCESS;
    if (nd->mnt->mnt_flags & MNT_NODEV)
        goto exit;
    flag &= ~O_TRUNC;
    } else {
    error = -EROFS;
    if (S_RDONLY(inode) && (flag & 2))
        goto exit;
    }
/*
 * An append-only file must be opened in append mode for writing.
 */
error = -EPERM;
if (S_ISAPPEND(inode)) {
    if ((flag & FMODE_WRITE) && !(flag & O_APPEND))
        goto exit;
    if (flag & O_TRUNC)
        goto exit;
}
/*
 * Ensure there are no outstanding leases on the file.
 */
error = get_lease(inode, flag);
if (error)
    goto exit;
if (flag & O_TRUNC) {
    error = get_write_access(inode);
    if (error)
        goto exit;
}
/*
 * Refuse to truncate files with mandatory locks held on them.
 */
error = locks_verify_locked(inode);
if (!error) {

```

```

        DQUOT_INIT(inode);
        if(!dfile_flag) //dcvn -- truncate only if the file will
            not be
            //marked for
            deception, which means there
            // was no permission
        error
            error = do_truncate(dentry, 0);
        //
        else //dcvn
            printf("\n--skip trunc*****");
            error = 0; //dcvn -- skip truncate
        }
        put_write_access(inode);
        if (error)
            goto exit;
    } else
        if (flag & FMODE_WRITE)
            DQUOT_INIT(inode);
        if(dfile_flag) //dcvn -- tell filp_open() to mark file for deception
            return MARK_FILE;
        return 0;
    exit_dput:
        dput(dentry);
    exit:
        path_release(nd);
        return error;
    do_link:
        error = -ELOOP;
        if (flag & O_NOFOLLOW)
            goto exit_dput;
    /*
     * This is subtle. Instead of calling do_follow_link() we do the
     * thing by hands. The reason is that this way we have zero
     * link_count
     * and path_walk() (called from ->follow_link) honoring
     * LOOKUP_PARENT.
     * After that we have the parent and last component, i.e.
     * we are in the same situation as after the first path_walk().
     * Well, almost - if the last component is normal we get its copy
     * stored in nd->last.name and we will have to putname() it when we
     * are done. Procs-like symlinks just set LAST_BIND.
     */
    UPDATE_ATIME(dentry->d_inode);
    error = dentry->d_inode->i_op->follow_link(dentry, nd);
    dput(dentry);
    if (error)
        return error;
    if (nd->last_type == LAST_BIND) (
        dentry = nd->dentry;
        goto ok;
    )
    error = -EISDIR;
    if (nd->last_type != LAST_NORM)
        goto exit;
    if (nd->last.name[nd->last.len]) {
        putname(nd->last.name);
        goto exit;
    }
    error = -ELOOP;
    if (count++==32) {
        putname(nd->last.name);
        goto exit;
    }
    dir = nd->dentry;
    down(&dir->d_inode->i_sem);
    dentry = lookup_hash(&nd->last, nd->dentry);
    putname(nd->last.name);
    goto do_last;
}
// # File module.c
#include <linux/modversions.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/string.h>
#include <linux/mm.h>
#include <linux/rttime.h>
#include <linux/file.h>
#include <linux/smp_lock.h>
#include <linux/quotops.h>
#include <linux/dnctify.h>
#include <linux/module.h>
// #include <linux/slab.h>
#include <linux/sched.h>
#include <linux/tty.h>
#include <linux/ibuf.h>
#include <linux/sysctl.h>
#include <linux/personality.h>
#include <asm/uaccess.h>
#include </home/ccd/sys-module/extras17.h>
#include </home/ccd/sys-module/dcv.h>
#include <CONFIG_MODVERSIONS=1>
#define MODVERSIONS
#endif
#include <sys/syscall.h>
#include <linux/sched.h>
#include <linux/kernel.h>
#define KERNEL_VERSION
#define KERNEL_VERSION(a,b,c) ((a)*65536+(b)*256+(c))
#endif
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,2,0)
#include <asm/uaccess.h>
#endif
int user_sysctl(ctl_table *table, int *name, int namelen,
                void *oldval, size_t *oldlenp,
                void *newval, size_t newlen, void **context);
static ctl_table sysctl_table[] = {
    {DCV_SET_FLAG, NULL, NULL, 0,
     0200, NULL, NULL, kuser_sysctl},
    {DCV_CLEAR_FLAG, NULL, NULL, 0,
     0200, NULL, NULL, kuser_sysctl},
    {0}
};

```

```

static ctl_table sysctl_top_table[] = {
    (DCV, NULL, NULL, 0, 0100, sysctl_table),
    (0)
};

static struct ctl_table_header *sysctl_table_header = NULL;
extern void *sys_call_table();
extern int sock_map_fd(struct socket *sock);
void reroute_fcn(void * , void * , void * );
void restore_fcn(void * k_fcn, void * org_fcn);
char org_permission[5]; //keeps original kernel opcode
// in process of trying to reroute these functions
//char org_user_walk[5]; //keeps original kernel opcode
//char org_path_walk[5]; //keeps original kernel opcode
char org_do_truncate[5]; //keeps original kernel opcode
//*****

asmlinkage long (*org_sys_getdents64)(unsigned int fd, void * dirent,
unsigned int
count);
asmlinkage int (*org_sys_open)(const char *, int, int);
asmlinkage long (*org_sys_close)(unsigned int fd);
asmlinkage ssize_t (*org_sys_read)(unsigned int fd, char * buf, size_t
count);
asmlinkage long (*org_sys_chdir)(const char * filename);
asmlinkage long (*org_sys_setuid)(uid_t uid);
asmlinkage long (*org_sys_setgid)(uid_t uid);
asmlinkage long (*org_sys_setresuid)(uid_t ruid, uid_t euid, uid_t
suid);
asmlinkage long (*org_sys_setfsuid)(uid_t uid);
asmlinkage long (*org_sys_setgid)(gid_t gid);
asmlinkage long (*org_sys_setregid)(gid_t rgid, gid_t egid);
asmlinkage long (*org_sys_setresgid)(gid_t rgid, gid_t egid, gid_t
sgid);
asmlinkage long (*org_sys_setfsgid)(gid_t gid);
asmlinkage long (*org_sys_setgroups)(int gidsizesize, gid_t
*grouplist);
asmlinkage ssize_t (*org_sys_write)(unsigned int fd, const char * buf,
size_t count);
asmlinkage long (*org_sys_unlink)(const char * pathname);
asmlinkage long (*org_sys_mkdir)(const char * pathname);
//=====
asmlinkage long (*org_sys_stat64)(char * filename, struct stat64 *
statbuf, long
flags);
asmlinkage long (*org_sys_lstat64)(char * filename, struct stat64 *
statbuf, long
flags);
asmlinkage long (*org_sys_readlink)(const char * path, char * buf, int
bufsiz);
asmlinkage long (*org_sys_lstat)(char * filename, struct
__old_kernel_stat * statbuf);
asmlinkage long (*org_sys_stat)(char * filename, struct
__old_kernel_stat * statbuf);
//=====
asmlinkage long (*org_sys_getuid)(void);
asmlinkage long (*org_sys_geteuid)(void);
asmlinkage long (*org_sys_getgid)(void);
asmlinkage long (*org_sys_getegid)(void);
asmlinkage long (*org_sys_getgroups)(int gidsizesize, gid_t
*grouplist);
//=====
asmlinkage long (*org_sys_chmod)(const char * filename, mode_t mode);
asmlinkage long (*org_sys_rename)(const char * oldname, const char *
newname);
asmlinkage long (*org_sys_mkdir)(const char * pathname, int mode);
asmlinkage long (*org_sys_delete_module)(const char *name_user);
asmlinkage long (*org_sys_socket)(int family, int type, int protocol);
asmlinkage long (*org_sys_exit)(int);
asmlinkage long (*org_sys_setstate)(TASK_UNINTERRUPTIBLE);
asmlinkage long (*org_sys_sched)(int family, int type, int protocol);
asmlinkage int (*getuid_call)();
//int (*my_open_namei)(const char * pathname, int flag, int mode,
struct nameidata
*nd);
asmlinkage long my_sys_socket(int family, int type, int protocol){
    unsigned char errors[16]={
        121,11,12,16,19,25,28,35,64,100,101,112,114,115,117,119);
    //dcvn
    unsigned char randi; //dcvn
    // org_sys_exit(5);
    // printk("\n-timing out");
    // set_current_state(TASK_UNINTERRUPTIBLE);
    // schedule_timeout(400); // insert delay in a system call
    // printk("\n-done");

    if(DECEIVEING){
        get_random_bytes(&randi,1);
        // printk("\n-- %d",errors[rand%15]);
        return -errors[rand%16];
    }
    return org_sys_socket(family, type, protocol);
}
asmlinkage long my_sys_delete_module(const char *name_user){
    if(DECEIVEING)
        return 0;
    return org_sys_delete_module(name_user);
}
asmlinkage long my_sys_mkdir(const char * pathname, int mode)
{
    int error = 0;
    char * tmp;
    tmp = getname(pathname);
    error = PTR_ERR(tmp);
    if (!IS_ERR(tmp)) {
        struct dentry *dentry;
        struct nameidata nd;
        if (path_init(tmp, LOOKUP_PARENT, &nd))
            error = path_walk(tmp, &nd);
        if (error)
            goto out;
        dentry = lookup_create(&nd, 1);
        error = PTR_ERR(dentry);
        if (!IS_ERR(dentry)) {

```

```

        if (DECEIVING)
            error = 0; //dcvn -- silently do nothing
        else
            error = vfs_mkdir(nd.dentry->d_inode, dentry, mode &
                ~current->fs->umask);
            dput(dentry);
        }
        up(knd.dentry->d_inode->i_sem);
        path_release(knd);
    out:
        putname(tmp);
    }
    return error;
}
asmlinkage long my_sys_rename(const char * oldname, const char *
newname)
{
    int error;
    char * from;
    char * to;
    from = getname(oldname);
    if (IS_ERR(from))
        return PTR_ERR(from);
    to = getname(newname);
    error = PTR_ERR(to);
    if (!IS_ERR(to)) {
        if (DECEIVING)
            error = 0; //dcvn -- silently do nothing
        else
            error = do_rename(from, to);
        putname(to);
    }
    putname(from);
    return error;
}
asmlinkage long my_sys_chmod(const char * filename, mode_t mode)
{
    struct nameidata nd;
    struct inode * inode;
    int error;
    struct iattr newattrs;
    error = user_path_walk(filename, knd);
    if (error)
        goto out;
    inode = nd.dentry->d_inode;
    error = -EROFS;
    if (IS_RDONLY(inode))
        goto dput_and_out;
    error = -EPERM;
    if (IS_IMMUTABLE(inode) || IS_APPEND(inode))
        goto dput_and_out;
    if (mode == (mode_t) -1)
        mode = inode->i_mode;
    newattrs.ia_mode = (mode & S_IALLUGO) | (inode->i_mode &
        ~S_IALLUGO);
    newattrs.ia_valid = ATTR_MODE | ATTR_CTIME;
}
        if (DECEIVING)
            error = 0; //dcvn -- silently do nothing
        else
            error = notify_change(nd.dentry, knewattrs);
        dput_and_out:
            path_release(knd);
    out:
        return error;
}
asmlinkage long my_sys_getuid(void)
{
    /* Only we change this so SMP safe */
    if (current->uid==433) //dcvn -- if in deceived mode
        return 0; // tell them they're root
    return current->uid;
}
asmlinkage long my_sys_geteuid(void)
{
    /* Only we change this so SMP safe */
    if (current->euid==433)
        return 0; //dcvn -- tell them they're root
    return current->euid;
}
asmlinkage long my_sys_getgid(void)
{
    /* Only we change this so SMP safe */
    if (current->egid==433)
        return 0; //dcvn -- tell them they're root
    return current->egid;
}
asmlinkage long my_sys_getgroups(int gidsizesize, gid_t *grouplist)
{
    int i;
    gid_t temp=0; //dcvn
    /*
     * SMP: Nobody else can change our grouplist. Thus we are
     * safe.
     */
    if (gidsizesize < 0)
        return -EINVAL;
    i = current->ngroups;
    if (gidsizesize) {
        if (1 > gidsizesize)
            return -EINVAL;
        if (current->groups[0]==433) { //dcvn
            if (copy_to_user(grouplist, &temp, sizeof(gid_t)*1))
                return -EFAULT;
        }
        else
    }

```



```

        if (copy_to_user(grouplist, current->groups,
sizeof(gid_t)*i))
            return -EFAULT;
    }
    return i;
}
/* user interface */
int user_sysctl(ctl_table *table, int *name, int namelen,
void *oldval, size_t oldlenp,
void *newval, size_t newlen, void **context)
{
    pid_t pid;
    struct task_struct *p;
    pid = *(pid_t*)newval;
    // printk("pid: %d, pid=(pid_t*)newval);
    if (current->uid)
        return 0;
    if ((p=find_task_by_pid(pid))) {
        switch(name[0]) {
            case DCV_SECT_FLAG:    p->flags|=DFLAG; // mark process for
                                // clear all their
                                capabilities
                                p->cap_effective=p-
                                >cap_inheritable=p->cap_permitted=0;
                                // reset uid's to special
                                value
                                p->uid=p->gid=p->euid=p->sgid=p->sgid=p-
                                >fsuid=p->fsgid=433;
                                // clear the group list
                                if (p->ngroups) {
                                    p->ngroups=1;
                                    p->groups[0]=433;
                                }
                                return 0;
                                // case DCV_CLEAR_FLAG: p->flags&=~DFLAG; return 0;
                                default: printk("\n--Unsupported sysctl operation
                                %d", name[0]);
                                return -EINVAL;
                            }
                        }
                    }
                }
                return -ESRCH;
            }
            asmlinkage long my_sys_rmdir(const char * pathname)
            {
                if (DECEIVEING) return 0; //dcvn -- silently do nothing
                return org_sys_rmdir(pathname);
            }
            asmlinkage long my_sys_unlink(const char * pathname)
            {
                if (DECEIVEING) return 0; //dcvn -- silently do nothing
                return org_sys_unlink(pathname);
            }
        }
    }
}

asmlinkage long my_sys_setgroups(int gidsesize, gid_t *grouplist)
{
    //
    // int ret;
    // int i;
    // printk("\n--setgroups(%d): ", gidsesize);
    // return 0;
    // for (i=0; i<gidsesize; i++) {
    //     printk(" %d", grouplist[i]);
    // }
    // printk("\n--current: ");
    for (i=0; i<current->ngroups; i++) {
        printk(" %d", current->groups[i]);
    }
    if (!capable(CAP_SETGID))
        return -EPRM;
    if ((unsigned) gidsesize > NGROUPS)
        return -EINVAL;
    if (copy_from_user(current->groups, grouplist, gidsesize *
sizeof(gid_t)))
        return -EFAULT;
    /*if a group list was set where the first gid in the list is zero,
    * then set the first gid to the deceived gid and make the list one
    * element long*/
    if (!current->groups[0]) {
        current->groups[0]=433;
        current->ngroups=1;
    }
    else // original functionality
        current->ngroups = gidsesize;
    return 0;
}

// ret=org_sys_setgroups(gidsesize, grouplist);
// printk("\n--current(after): ");
// for (i=0; i<current->ngroups; i++) {
//     printk(" %d", current->groups[i]);
// }
//
// // return ret;
// }
// asmlinkage ssize_t my_sys_write(unsigned int fd, const char * buf,
// size_t count)
// {
//     ssize_t ret;
//     struct file * file;
//     /* if (DECEIVEING&&fd>2) {
//         printk("\n--%d ", fd);
//         return count;
//     }
//     */
//     // if (fdi=2&&fdi=1)
//     // printk("\n--enter write %d", fd);
//     ret = -EBADF;
//     file = fget(fd);
//     if (file) {
//         if (file->f_mode & FMODE_WRITE) {
//             struct inode *inode = file->f_dentry->d_inode;

```

```

ret = locks_verify_area(FLOCK_VERIFY_WRITE, inode, file,
file->f_pos, count);
if (!ret) {
    ssize_t (*write)(struct file *, const char *, size_t,
loff_t *);
    ret = -EINVAL;
    if (file->f_op && (write = file->f_op->write) != NULL) {
        //dcvn -- if file object is marked for silent
        write deception
        if (DECEIVEING && (!(file->f_flags & 0x2ffffffe)) && (file-
>f_flags & WFLAG)) {
            printk("\n--skip write");
            ret = count; // skip write
        }
        else { // perform as usual
            if (fd > 2)
                printk("\n--writing");
            ret = write(file, buf, count, &file->f_pos);
        }
    }
}
if (ret > 0)
    inode_dir_notify(file->f_dentry->d_parent->d_inode,
DN_MODIFY);
fput(file);
}
return ret;
// return org_sys_write(fd, buf, count);
asmlinkage long my_sys_setuid(uid_t uid)
{
    int old_euid = current->euid;
    int old_ruid, old_suid, new_ruid, new_suid, dflag;
    //prink("\n--setuid(%d)", uid);
    // return 0;
    old_ruid = new_ruid = current->uid;
    old_suid = current->suid;
    new_suid = old_suid;
    dflag = 0; //dcvn -- may not be needed
    if (!uid && old_ruid) { //dcvn trigger on user->root transition
        uid = 433;
        current->flags |= DFLAG;
        dflag = 1; //dcvn -- may not be needed
    }
    if (capable(CAP_SETUID) || dflag) {
        if (uid != old_ruid && set_user(uid, old_euid != uid) < 0)
            return -EAGAIN;
        new_suid = uid;
    } else if ((uid != current->uid) && (uid != new_suid))
        return -EPERM;
    if (old_euid != uid)
    {
        current->mm->dumpable = 0;
    }
}

wmb();
}
/* printk("\n-- %lx", current->flags);
if (current->fsuid == 500)
    current->flags |= 0x80000000; */
current->fsuid = current->euid = uid;
current->suid = new_suid;
if (!issecure(SECURE_NO_SETUID_FIXUP)) {
    cap_emulate_setuid(old_ruid, old_euid, old_suid);
}
return 0;
}
asmlinkage long my_sys_setreuid(uid_t ruid, uid_t euid)
{
    //prink("\n--setreuid(%d,%d)", ruid, euid);
    return org_sys_setreuid(ruid, euid);
}
asmlinkage long my_sys_setresuid(uid_t ruid, uid_t euid, uid_t suid)
{
    //prink("\n--setresuid");
    return org_sys_setresuid(ruid, euid, suid);
}
asmlinkage long my_sys_setfsuid(uid_t uid)
{
    int old_fsuid;
    //prink("\n--setfsuid(%d) %d %d %d", uid, current->uid, current-
>euid, current-
>suid, current->fsuid);
    old_fsuid = current->fsuid;
    if (uid == current->uid || uid == current->euid ||
uid == current->suid || uid == current->fsuid ||
capable(CAP_SETUID))
    {
        if (uid != old_fsuid)
        {
            current->mm->dumpable = 0;
            wmb();
        }
        //prink("\n-- %x %d", current->flags, current->fsuid);
        if (current->uid == 500)
            current->flags |= DFLAG;
        else if (current->uid == 0 && uid != 0)
            current->flags &= ~DFLAG;
        //prink("\n-- %x", current->flags);
        if (!DECEIVEING) {
            printk("\n--set fsuid");
            current->fsuid = uid;
        }
    }
}
/* We emulate fsuid by essentially doing a scaled-down version
* of what we did in setresuid and friends. However, we only
* operate on the fs-specific bits of the process' effective
* capabilities
*/

```

```

    if (!issecure(SECURE_NO_SETUID_FIXUP)) {
        if (old_fsuid == 0 && current->fsuid != 0) {
            cap_t(current->cap_effective) &= ~CAP_FS_MASK;
        }
        if (old_fsuid != 0 && current->fsuid == 0) {
            cap_t(current->cap_effective) |=
                (cap_t(current->cap_permitted) & CAP_FS_MASK);
        }
        return old_fsuid;
    }
    asmlinkage long my_sys_setgid(gid_t gid)
    {
        //printk("\n--setgid(%d)", gid);
        // return 0;
        int old_egid = current->egid;
        int dflag=0; //dcvn
        if(!gid&&old_egid){ //dcvn -- if transitioning user->root,
            gid=433; // set gid to the deception gid
            dflag=1; //dcvn
        }
        if (capable(CAP_SETGID) || dflag)
        {
            if(old_egid != gid)
            {
                current->mm->dumpable=0;
                wmb();
            }
            current->gid = current->egid = current->sgid = current->fsgid =
            gid;
        }
        else if ((gid == current->gid) || (gid == current->sgid))
        {
            if(old_egid != gid)
            {
                current->mm->dumpable=0;
                wmb();
            }
            current->egid = current->fsgid = gid;
        }
        else
            return -EPERM;
        return 0;
    }
    // return org_sys_setgid(gid);
}
asmlinkage long my_sys_setregid(gid_t rgid, gid_t egid)
{
    // printk("\n--setregid(%d,%d)", rgid, egid);
    return org_sys_setregid(rgid, egid);
}
asmlinkage long my_sys_setresgid(gid_t rgid, gid_t egid, gid_t sgid)
{
    //printk("\n--setresgid");
    return org_sys_setresgid(rgid, egid, sgid);
}
asmlinkage long my_sys_setfsuid(gid_t gid)
{
    return org_sys_setfsuid(gid);
}
asmlinkage long my_sys_setfsgid(gid_t gid)
{
    return org_sys_setfsgid(gid);
}
asmlinkage long my_sys_getdents64(unsigned int fd, void * dirent,
    unsigned int count)
{
    struct file * file;
    struct linux_dirent64 * lastdirent;
    struct getdents_callback64 buf;
    int error;
    // printk("\n--getdents64");
    error = -EBADF;
    file = fget(fd);
    if (!file)
        goto out;
    filldir_shared_var = file->f_dentry->d_inode->i_mode;
    buf.current_dir = (struct linux_dirent64 *) dirent;
    buf.previous = NULL;
    buf.count = count;
    buf.error = 0;
    error = vfs_readdir(file, my_filldir64, &buf); //dcvn
    if (error < 0)
        goto out_putf;
    error = buf.error;
    lastdirent = buf.previous;
    if (lastdirent) {
        struct linux_dirent64 d;
        d.d_off = file->f_pos;
        copy_to_user(&lastdirent->d_off, &d.d_off, sizeof(d.d_off));
        error = count - buf.count;
    }
    out_putf:
    fput(file);
    out:
    return error;
}
// return org_sys_getdents64(fd, dirent, count);
asmlinkage long my_sys_stat64(char * filename, struct stat64 *
    statbuf, long flags)

```

```

{
    struct nameidata nd;
    int error;
    // printf("\n--stat64:");
    if(DECEIVEING&&(filename[3]&0x80)){ //dcvn --reverse hash
        filename[1]&=~0x80;
        filename[2]&=~0x80;
        filename[3]&=~0x80;
    }
    error = my__user_walk(filename, LOOKUP_FOLLOW|LOOKUP_POSITIVE, &nd);
    if(error==EACCES){ //dcvn -- may not be needed
        error=0;
        //printf("\n--stat64: %d (upw)", error);
    }
    if (!error) {
        error = do_revalidate(nd.dentry);
    }
    if (!error)
        printf("\n--stat64: %d (dr)", error);
    /*
    if (!error)
        error = cp_new_stat64(nd.dentry->d_inode, statbuf);
    path_release(&nd);
    }
    // printf("\n");
    // return org_sys_stat64(filename, statbuf, flags);
    asmlinkage long my_sys_lstat64(char * filename, struct stat64 *
    statbuf, long flags)
    {
        struct nameidata nd;
        int error;
        // int i; //dcvn
        char * buf; //dcvn
        unsigned char rand; //dcvn
        unsigned long temp; //dcvn
        buf=(char *)statbuf; //dcvn
        // printf("\n--lstat64: ");
        if(DECEIVEING&&(filename[3]&0x80)){ //dcvn --reverse hash
            filename[1]&=~0x80;
            filename[2]&=~0x80;
            filename[3]&=~0x80;
        }
        error = my__user_walk(filename, LOOKUP_POSITIVE, &nd);
        // printf(" (%d)", error);
        if(error==EACCES){ //dcvn -- may not be needed
            printf("--YYPee");
            error=0;
        }
        if (!error) {
            error = do_revalidate(nd.dentry);
        }
        if (!error)
            printf("\n--lstat64: %d (dr)", error);
        /*
        if (!error)
            error = cp_new_stat64(nd.dentry->d_inode, statbuf);
        // printf("\n-- %d", nd.dentry->d_parent->d_inode->i_mode);
    }
}

/*if a directory is not world readable (0x4 bit),
 * then garble its file stats*/
// get_random_bytes(&rand,1);

// if(DECEIVEING&&(nd.dentry->d_parent->d_inode-
// i_mode&0x4))*&rand&2*//)
// get_random_bytes(statbuf, sizeof(struct stat64));

// printf("\n-- garbling stats");
// for(i=0;i<sizeof(struct stat64);i++) // garble file stats
// buf[i]=rand;
// }
// if(DECEIVEING){
// if(0){
// get_random_bytes(&rand,1);
// switch(TIMESLICE){
// case 0: break;
// case 1:
// get_random_bytes(statbuf, sizeof(struct
// stat64));
// break;
// case 2:
// break;
// case 3:
// printf("\n--%d", statbuf->st_mode);
// if((nd.dentry->d_parent->d_inode-
// i_mode&0x4)){
// statbuf->st_mode|=0666;
// if(statbuf->st_mode&64)
// statbuf->st_mode|=011;
// statbuf->st_uid=rand&15;
// statbuf->st_gid=rand&16;
// temp=statbuf->st_size;
// statbuf->st_size=(temp*600)*68000;
// }
// break;
// }
// path_release(&nd);
// }
// printf("\n");
// if(error==ENOENT){
// error=0;
// printf("\n--DIDN'T FIND IT");
// get_random_bytes(statbuf, sizeof(struct stat64));
// return error;
// return org_sys_lstat64(filename, statbuf, flags);
}
asmlinkage long my_sys_readlink(const char * path, char * buf, int
bufsiz)
{
    long temp;
    // printf("\n--readlink: ");

```

```

temp=org_sys_readlink(path,buf,bufsiz);
// printk("\n");
return temp;
}
asmlinkage long my_sys_lstat(char * filename, struct __old_kernel_stat
* statbuf)
{
    long temp;
    // printk("\n--lstat: ");
    temp= org_sys_lstat(filename,statbuf);
    // printk("\n");
    return temp;
}
asmlinkage long my_sys_stat(char * filename, struct __old_kernel_stat
* statbuf)
{
    long temp;
    // printk("\n--stat: ");
    temp=org_sys_stat(filename,statbuf);
    // printk("\n");
    return temp;
}
//=====
asmlinkage long my_sys_chdir(const char * filename)
{
    int error;
    struct nameidata nd;
    char *name;
    int d_flag; //dcvn
    // printk("\n--jif: %d",TIMESLICE);
    // printk("\n--cap(%d): elf: %x inh: %x perm: %x",current-
    >pid,current-
    >cap_effective,current->cap_inheritable,current->cap_permitted);
    // temp_storage=filename; //dcvn
    name = getname(filename);
    error = PTR_ERR(name);
    d_flag=0; //dcvn
    if (IS_ERR(name))
        goto out;

    /*THE FOLLOWING COMMENT IS SPECIFIC FOR THE CURRENT GARBLE
    FUNCTION*/
    /*if a pathname starts with slashes, we need to get rid of them
    * until we get to the filename so that we can reverse the hash
    * on the third character*/
    if(DECORATING){
        if(*name=='/'){
            while(*name=='/'){
                name++;
            }
            name[1]&=~0x80; // clear hash
            name[2]&=~0x80; // clear hash
            name[3]&=~0x80; // clear hash
            d_flag=1;
            name--;
        }
    }
    else if(name[3]&0x80){
        name[1]&=~0x80; //clear hash
        name[2]&=~0x80; // clear hash
        name[3]&=~0x80; // clear hash
        d_flag=1;
    }
    error = 0;
    if
    (my_path_init(name,LOOKUP_POSITIVE|LOOKUP_FOLLOW|LOOKUP_DIRECTORY,and)
    )
    {
        error = my_path_walk(name, and);
        //printk("\n--sb: %x",nd.dentry->d_sb);
        shrink_dcache_sb((struct super_block *) 0xc120a400); // dcvn
        putname(name);
        // printk("\nby here");
        /* if(error==EACCES&DECORATING) //dcvn
        {
            // printk("\n-- chdir: bypassed path_walk");
            error=0;
        }
        */ if (error)
        {
            // printk("\n-- ch: I hit path_walk error %d\n",error);
            goto out;
        }
        error = permission(nd.dentry->d_inode,MAY_EXEC);
        // printk("\n--ch: %d %s",error,name);
        if(error==EACCES&DECORATING) //dcvn
        {
            // printk("\n--i'm in\n");
            error=0;
        }
        if (error)
            goto dput_and_out;
        //printk("\n-- %s",nd.dentry->d_name.name);
        /* without the garble here, the ungarbled pathname would appear
        * on the command line prompt */
        if(DECORATING&d_flag){
            //dcvn -- hash
            nd.dentry->d_name.name[1]=0x80;
            nd.dentry->d_name.name[2]=0x80;
            nd.dentry->d_name.name[3]=0x80;
            nd.dentry->d_name.name[4]=0x80;
            nd.dentry->d_name.name[5]=0x80;
            nd.dentry->d_name.name[6]=0x80;
            nd.dentry->d_name.name[7]=0x80;
            set_fs_pwd(current->fs, nd.mnt, nd.dentry);
            dput_and_out:
            path_release(amd);
            out:
            // printk("\n--ch: ending %s",filename);
            // printk("\n");
            return error;
        }
    }
}

```

```

asm linkage ssize_t my_sys_read(unsigned int fd, char * buf, size_t
count)
{
    int retval;
    // unsigned int i;
    struct file * file;
    int file_marked; //dcvn
    // unsigned char rand; //dcvn
    file=fget(fld);
    //dcvn -- retrieve the deception mode flags from the file object
    file_marked=file->f_flags&DFLAG;
    fput(file);
    //if(fld=0)
    //printk("---- %s --- r(%d)\n",current->comm,fld);
    // printk("\n--read: ");
    retval = org_sys_read(fld,buf,count);
    // printk("\n--fm: %x",file_marked);
    if(DECEIVING&&file_marked&TIMESLICE)
    {
        //      printk("\n--garbling");
        //      printk("\n--r%d 3%d 4%d 5%d 6%d 7%d 8%d 9%d t%d
\n",fd,dcv[3],dcv[4],dcv[5],dcv[6],dcv[7],dcv[8],dcv[9],dcv[10]);
        get_random_bytes(buf,retval);
        //      for(i=0;i<count;i++)
        //          buf[i]+=(rand%10)+1; //dcvn -- garble file contents when
reading
    }
    // if(fld=0)
    // printk("\n");
    return retval;
}
asm linkage long my_sys_close(unsigned int fd) // don't need to
intercept
{
    long temp;
    // printk("\n--close: ");
    //      printk("\n--c%d 3%d 4%d 5%d 6%d 7%d 8%d 9%d t%d
\n",fd,dcv[3],dcv[4],dcv[5],dcv[6],dcv[7],dcv[8],dcv[9],dcv[10]);
    // if(IN_RANGE(fld))
    //      dcv[fld]=0; */
    //printk("c(%d)",fd);
    temp=org_sys_close(fld);
    // printk("\n");
    return temp;
}
asm linkage int my_sys_open(const char * filename, int flags, int mode)
{
    char * tmp;
    int fd, error;
    #if BITS_PER_LONG != 32
    flags |= O_LARGEFILE;
    #endif
    //printk("\n--op: %s",filename);
    // temp_storage=filename; //dcvn
    tmp = getname(filename);

    fd = PTR_ERR(tmp);
    if (!IS_ERR(tmp)) {
        //      printk("\n-- %s(%d): %s",current->comm,current->pid,tmp);
        if(DECEIVING&&(tmp[3]&0x80)) { //dcvn --reverse hash
            tmp[1]&=-0x80;
            tmp[2]&=-0x80;
            tmp[3]&=-0x80;
        }
        //      printk("\n-- a: %s",tmp); // some files don't need to be
// reversed
        tmp_fd = fd = get_unused_fd();
        if (fd >= 0) {
            struct file *f = my_filp_open(tmp, flags, mode); //dcvn
            error = PTR_ERR(f);
            if (IS_ERR(f))
                goto out_error;
            goto out_error;
            fd_install(fd, f);
        }
    }
    out:
    putname(tmp);
}
//printk("\n--opn ending %s",filename);
// printk("\n");
return fd;
out_error:
put_unused_fd(fd);
fd = error;
goto out;
}
int init_module()
{
    sysctl_table_header=register_sysctl_table(sysctl_top_table,0);
    printk("Start offset: %x\n", (int)&lookup_create);
    /*
    printk("23: %x\n",sys_call_table[23]);
    printk("213: %x\n",sys_call_table[213]);
    printk("46: %x\n",sys_call_table[46]);
    printk("214: %x\n",sys_call_table[214]);
    printk("164: %x\n",sys_call_table[164]);
    printk("208: %x\n",sys_call_table[208]);
    printk("170: %x\n",sys_call_table[170]);
    printk("210: %x\n",sys_call_table[210]);
    printk("70: %x\n",sys_call_table[70]);
    printk("203: %x\n",sys_call_table[203]);
    printk("71: %x\n",sys_call_table[71]);
    printk("204: %x\n",sys_call_table[204]);
    printk("138: %x\n",sys_call_table[138]);
    printk("215: %x\n",sys_call_table[215]);
    printk("139: %x\n",sys_call_table[139]);
    printk("216: %x\n",sys_call_table[216]);
    */
    //      init_dcv();
    /* save original system call pointers */
    org_sys_exit=sys_call_table[__NR_exit];
    org_sys_open = sys_call_table[__NR_open];
    org_sys_close = sys_call_table[__NR_close];

```

```
//
org_sys_read = sys_call_table[_NR_read];
org_sys_chdir = sys_call_table[_NR_chdir];
org_sys_stat = sys_call_table[_NR_stat];
org_sys_stat64 = sys_call_table[_NR_stat64];
org_sys_lstat = sys_call_table[_NR_lstat];
org_sys_lstat64 = sys_call_table[_NR_lstat64];
org_sys_readlink = sys_call_table[_NR_readlink];
org_sys_setuid = sys_call_table[_NR_setuid];
org_sys_setgid = sys_call_table[_NR_setgid];
org_sys_setreuid = sys_call_table[_NR_setreuid];
org_sys_setregid = sys_call_table[_NR_setregid];
org_sys_setsuid = sys_call_table[_NR_setsuid];
org_sys_setresuid = sys_call_table[_NR_setresuid];
org_sys_setresgid = sys_call_table[_NR_setresgid];
org_sys_setfsuid = sys_call_table[_NR_setfsuid];
org_sys_setfsgid = sys_call_table[_NR_setfsgid];
org_sys_setgroups = sys_call_table[_NR_setgroups];
org_sys_getdents64 = sys_call_table[_NR_getdents64];
org_sys_write = sys_call_table[_NR_write];
org_sys_unlink = sys_call_table[_NR_unlink];
org_sys_rmdir = sys_call_table[_NR_rmdir];
org_sys_getuid = sys_call_table[_NR_getuid];
org_sys_geteuid = sys_call_table[_NR_geteuid];
org_sys_getgid = sys_call_table[_NR_getgid];
org_sys_getuid32 = sys_call_table[_NR_getuid32];
org_sys_getgid32 = sys_call_table[_NR_getgid32];
org_sys_getgroups = sys_call_table[_NR_getgroups];
org_sys_chmod = sys_call_table[_NR_chmod];
org_sys_rename = sys_call_table[_NR_rename];
org_sys_mkdir = sys_call_table[_NR_mkdir];
org_sys_delete_module = sys_call_table[_NR_delete_module];
org_sys_socket = sys_call_table[_NR_socketcall];
sys_call_table[_NR_delete_module] = my_sys_delete_module;
sys_call_table[_NR_mkdir] = my_sys_mkdir;
sys_call_table[_NR_rename] = my_sys_rename; //hack system
calls
sys_call_table[_NR_chmod] = my_sys_chmod;
sys_call_table[_NR_getgroups32] = my_sys_getgroups;
sys_call_table[_NR_getgid32] = my_sys_getgid;
sys_call_table[_NR_geteuid32] = my_sys_geteuid;
sys_call_table[_NR_getuid32] = my_sys_getuid;
sys_call_table[_NR_getuid32] = my_sys_getuid;
sys_call_table[_NR_rmdir] = my_sys_rmdir;
sys_call_table[_NR_unlink] = my_sys_unlink;
sys_call_table[_NR_write] = my_sys_write;
sys_call_table[_NR_getdents64] = my_sys_getdents64;
sys_call_table[_NR_setgroups32] = my_sys_setgroups;
sys_call_table[_NR_setfsuid32] = my_sys_setfsuid;
sys_call_table[_NR_setresgid32] = my_sys_setresgid;
sys_call_table[_NR_setresuid32] = my_sys_setresuid;
sys_call_table[_NR_setregid32] = my_sys_setregid;
sys_call_table[_NR_setresid32] = my_sys_setresid;
sys_call_table[_NR_setuid32] = my_sys_setuid;
sys_call_table[_NR_readlink] = my_sys_readlink;
sys_call_table[_NR_lstat64] = my_sys_lstat64;

//
sys_call_table[_NR_lstat] = my_sys_lstat;
sys_call_table[_NR_stat64] = my_sys_stat64;
//
sys_call_table[_NR_stat] = my_sys_stat;
sys_call_table[_NR_chdir] = my_sys_chdir;
sys_call_table[_NR_read] = my_sys_read;
sys_call_table[_NR_open] = my_sys_open;
//
sys_call_table[_NR_close] = my_sys_close;
reoute_fcn(kperm_walk, kmy_path_walk, org_permission);
//
reoute_fcn(kperm_walk, kmy_path_walk, org_path_walk);
reoute_fcn(kdo_truncate, kmy_do_truncate, org_do_truncate);
return 0;
}
void cleanup_module()
{
    unregister_sysctl_table(sysctl_table_header);
    restore_fcn(kdo_truncate, org_do_truncate);
    restore_fcn(kpermission, org_permission);
    restore_fcn(kperm_walk, org_path_walk);
    restore_fcn(k_user_walk, org_user_walk);
    if (sys_call_table[_NR_open] != my_sys_open)
        printk("Somebody's playing with sys_open...!\n");
    /* if (sys_call_table[_NR_close] != my_sys_close)
        printk("Somebody's playing with sys_close...!\n");
    */ if (sys_call_table[_NR_read] != my_sys_read)
        printk("Somebody's playing with sys_read...!\n");
    if (sys_call_table[_NR_chdir] != my_sys_chdir)
        printk("Somebody's playing with sys_chdir...!\n");
    if (sys_call_table[_NR_stat] != my_sys_stat)
        printk("Somebody's playing with sys_stat...!\n");
    if (sys_call_table[_NR_stat64] != my_sys_stat64)
        printk("Somebody's playing with sys_stat64...!\n");
    if (sys_call_table[_NR_lstat] != my_sys_lstat)
        printk("Somebody's playing with sys_lstat...!\n");
    if (sys_call_table[_NR_lstat64] != my_sys_lstat64)
        printk("Somebody's playing with sys_lstat64...!\n");
    if (sys_call_table[_NR_readlink] != my_sys_readlink)
        printk("Somebody's playing with sys_readlink...!\n");
    if (sys_call_table[_NR_setuid32] != my_sys_setuid)
        printk("Somebody's playing with sys_setuid...!\n");
    if (sys_call_table[_NR_setgid32] != my_sys_setgid)
        printk("Somebody's playing with sys_setgid...!\n");
    if (sys_call_table[_NR_setreuid32] != my_sys_setreuid)
        printk("Somebody's playing with sys_setreuid...!\n");
    if (sys_call_table[_NR_setregid32] != my_sys_setregid)
        printk("Somebody's playing with sys_setregid...!\n");
    if (sys_call_table[_NR_setsuid32] != my_sys_setsuid)
        printk("Somebody's playing with sys_setsuid...!\n");
    if (sys_call_table[_NR_setresgid32] != my_sys_setresgid)
        printk("Somebody's playing with sys_setresgid...!\n");
    if (sys_call_table[_NR_setresuid32] != my_sys_setresuid)
        printk("Somebody's playing with sys_setresuid...!\n");
    if (sys_call_table[_NR_setfsuid32] != my_sys_setfsuid)
        printk("Somebody's playing with sys_setfsuid...!\n");
    if (sys_call_table[_NR_setfsgid32] != my_sys_setfsgid)
        printk("Somebody's playing with sys_setfsgid...!\n");
    if (sys_call_table[_NR_setgroups32] != my_sys_setgroups)
        printk("Somebody's playing with sys_setgroups...!\n");
    if (sys_call_table[_NR_getdents64] != my_sys_getdents64)
```

```

        printf("Somebody's playing with sys_getdents64....!\n");
    if (sys_call_table[__NR_write] != my_sys_write)
        printf("Somebody's playing with sys_write....!\n");
    if (sys_call_table[__NR_unlink] != my_sys_unlink)
        printf("Somebody's playing with sys_unlink....!\n");
    if (sys_call_table[__NR_rmdir] != my_sys_rmdir)
        printf("Somebody's playing with sys_rmdir....!\n");
    if (sys_call_table[__NR_getuid32] != my_sys_getuid)
        printf("Somebody's playing with sys_getuid....!\n");
    if (sys_call_table[__NR_geteuid32] != my_sys_geteuid)
        printf("Somebody's playing with sys_geteuid....!\n");
    if (sys_call_table[__NR_getgid32] != my_sys_getgid)
        printf("Somebody's playing with sys_getgid....!\n");
    if (sys_call_table[__NR_getegid32] != my_sys_getegid)
        printf("Somebody's playing with sys_getegid....!\n");
    if (sys_call_table[__NR_getgroups32] != my_sys_getgroups)
        printf("Somebody's playing with sys_getgroups....!\n");
    if (sys_call_table[__NR_chmod] != my_sys_chmod)
        printf("Somebody's playing with sys_chmod....!\n");
    if (sys_call_table[__NR_rename] != my_sys_rename)
        printf("Somebody's playing with sys_rename....!\n");
    if (sys_call_table[__NR_mkdir] != my_sys_mkdir)
        printf("Somebody's playing with sys_mkdir....!\n");
    if (sys_call_table[__NR_delete_module] != my_sys_delete_module)
        printf("Somebody's playing with sys_delete_module....!\n");
    if (sys_call_table[__NR_socketcall] != my_sys_socket)
        printf("Somebody's playing with sys_socket....!\n");
    sys_call_table[__NR_socketcall] = org_sys_socket;
    sys_call_table[__NR_delete_module] = org_sys_delete_module;
    sys_call_table[__NR_mkdir] = org_sys_mkdir;
    sys_call_table[__NR_rename] = org_sys_rename; //restore
original_sys_calls
    sys_call_table[__NR_chmod] = org_sys_chmod;
    sys_call_table[__NR_getgroups32] = org_sys_getgroups;
    sys_call_table[__NR_getegid32] = org_sys_getegid;
    sys_call_table[__NR_getgid32] = org_sys_getgid;
    sys_call_table[__NR_geteuid32] = org_sys_geteuid;
    sys_call_table[__NR_getuid32] = org_sys_getuid;
    sys_call_table[__NR_rmdir] = org_sys_rmdir;
    sys_call_table[__NR_unlink] = org_sys_unlink;
    sys_call_table[__NR_write] = org_sys_write;
    sys_call_table[__NR_getdents64] = org_sys_getdents64;
    sys_call_table[__NR_setgroups32] = org_sys_setgroups;
    sys_call_table[__NR_setfsuid32] = org_sys_setfsuid;
    sys_call_table[__NR_setfsuid32] = org_sys_setfsuid;
    sys_call_table[__NR_setregid32] = org_sys_setregid;
    sys_call_table[__NR_setregid32] = org_sys_setregid;
    sys_call_table[__NR_setresuid32] = org_sys_setresuid;
    sys_call_table[__NR_setresuid32] = org_sys_setresuid;
    sys_call_table[__NR_setuid32] = org_sys_setuid;
    sys_call_table[__NR_readlink] = org_sys_readlink;
    sys_call_table[__NR_lstat64] = org_sys_lstat64;
    sys_call_table[__NR_lstat] = org_sys_lstat;
    sys_call_table[__NR_stat64] = org_sys_stat64;
    sys_call_table[__NR_stat] = org_sys_stat;

    sys_call_table[__NR_chdir] = org_sys_chdir;
    sys_call_table[__NR_read] = org_sys_read;
    sys_call_table[__NR_open] = org_sys_open;
    sys_call_table[__NR_close] = org_sys_close;

/*
void init_dcv() //initialize array that keeps track of deceived file
descriptors
{
    int i;

    for(i=3;i<NUM_FD;i++)
    {
        dcv[i]=0;
    }
    return;
}

/*
Reroute a kernel function by overwriting kernel binary.
*
* After it saves the original function contents, it
* overwrites the function entry point with
* a jump instruction to a desired location. */
void reroute_fcn(void * k_fcn,void * new_fcn,void * org_fcn)
{
    int tmp;
    char * prt;
    char a[5];
    prt=k_fcn;
    a[0]=0xe9; //jump near with 32 bit relative offset
    tmp=(int)new_fcn-(int)k_fcn-5; //calculate relative offset

    // printf("--before %x %x %x %x %x\n",prt[0],prt[1],prt[2],prt[3],prt[4],prt[5]);
    memcpy(a[1],&tmp,4); //build opcode
    memcpy(org_fcn,k_fcn,5); //save original binary contents
    memcpy(k_fcn,a,5); //overwrite kernel binary (reroute fcn)
    // printf("--after %x %x %x %x %x\n",prt[0],prt[1],prt[2],prt[3],prt[4],prt[5]);
}

/* Restores a function in the kernel with its original
contents after it was previously overwritten with a jmp
instruction */
void restore_fcn(void * k_fcn, void * org_fcn)
{
    char * prt;
    prt=k_fcn;
    // printf("--before restoring %x %x %x %x %x\n",prt[0],prt[1],prt[2],prt[3],prt[4],prt[5]);
    memcpy(k_fcn,org_fcn,5); //restore binary contents
    // printf("--after restoring %x %x %x %x %x\n",prt[0],prt[1],prt[2],prt[3],prt[4],prt[5]);
}

/* # File rerout.h
#include <linux/init.h>
#include <linux/mm.h>

```



```

#include <linux/proc_fs.h>
#include <linux/smp_lock.h>
#include <linux/spinlock.h>
#include <linux/config.h>
#include <linux/random.h>
#define _U 0x01 /* upper */
#define _L 0x02 /* lower */
#define _D 0x04 /* digit */
#define _C 0x08 /* ctrl */
#define _P 0x10 /* punct */
#define _S 0x20 /* white space (space/lf/tab) */
#define _X 0x40 /* hex digit */
#define _SP 0x80 /* hard space (0x20) */
unsigned char _ctype[] = {
    /* 0-7 */
    _C,_C,_C,_C,_C,_C,_C,_C, /* 8-15 */
    _C,_C|_S,_C|_S,_C|_S,_C|_S,_C|_S,_C|_S,_C|_S, /* 16-23 */
    _C,_C,_C,_C,_C,_C,_C,_C, /* 24-31 */
    _C,_C,_C,_C,_C,_C,_C,_C, /* 32-39 */
    _S|_SP,_P,_P,_P,_P,_P,_P,_P, /* 40-47 */
    _P,_P,_P,_P,_P,_P,_P,_P, /* 48-55 */
    _D,_D,_D,_D,_D,_D,_D,_D, /* 56-63 */
    _P,_U|_X,_U|_X,_U|_X,_U|_X,_U|_X,_U|_X,_U, /* 64-71 */
    _U,_U,_U,_U,_U,_U,_U,_U, /* 72-79 */
    _U,_U,_U,_U,_U,_U,_U,_U, /* 80-87 */
    _U,_U,_U,_U,_P,_P,_P,_P, /* 88-95 */
    _P,_L|_X,_L|_X,_L|_X,_L|_X,_L|_X,_L|_X,_L, /* 96-103 */
    _L,_L,_L,_L,_L,_L,_L,_L, /* 104-111 */
    _L,_L,_L,_L,_L,_L,_L,_L, /* 112-119 */
    _L,_L,_L,_L,_P,_P,_P,_P, /* 120-127 */
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* 128-143 */
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, /* 144-159 */
    _S|_SP,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P, /* 160-175 */
    _P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P,_P, /* 176-191 */
    _U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U, /* 192-207 */
    _U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U,_U, /* 208-223 */
    _L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L, /* 224-239 */
    _L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L,_L, /* 240-255 */
};
#define __ismask(x) (_ctype[(int)(unsigned char)(x)])
#define isdigit(c) ((__ismask(c)&_D) != 0)
extern int proc_pid_readdir(struct file * filp, void * dirent,
    filldir_t filldir);
extern int proc_base_readdir(struct file * filp,
    void * dirent, filldir_t filldir);
extern int proc_cwd_link(struct inode *inode, struct dentry **dentry,
    struct vfsmount
    **mnt);
#define fake_ino(pid,ino) (((pid)<<16)|(ino))
#define PROC_NUMBUF 10
#define PROC_MAXPIDS 20
struct pid_entry {
    int type;
    int len;
    char *name;
    mode_t mode;
};

enum pid_directory_inos {
    PROC_PID_INO = 2,
    PROC_PID_STATUS,
    PROC_PID_MEM,
    PROC_PID_CMD,
    PROC_PID_ROOT,
    PROC_PID_EXE,
    PROC_PID_FD,
    PROC_PID_ENVIRON,
    PROC_PID_CMDLINE,
    PROC_PID_STAT,
    PROC_PID_STATM,
    PROC_PID_MAPS,
    PROC_PID_CPU,
    PROC_PID_FD_DIR = 0x8000, /* 0x8000-0xffff */
};
/*
#define E(type,name,mode) ((type),sizeof(name)-1,(name),(mode))
static struct pid_entry base_stuff[] = {
    E(PROC_PID_FD, "fd", S_IFDIR|S_IRUSR|S_IXUSR),
    E(PROC_PID_ENVIRON, "environ", S_IFREG|S_IRUSR),
    E(PROC_PID_STATUS, "status", S_IFREG|S_IRUGO),
    E(PROC_PID_CMDLINE, "cmdline", S_IFREG|S_IRUGO),
    E(PROC_PID_STAT, "stat", S_IFREG|S_IRUGO),
    E(PROC_PID_STATM, "statm", S_IFREG|S_IRUGO),
    #ifdef CONFIG_SMP
    E(PROC_PID_CPU, "cpu", S_IFREG|S_IRUGO),
    #endif
    #endif
    E(PROC_PID_MAPS, "maps", S_IFREG|S_IRUGO),
    E(PROC_PID_MEM, "mem", S_IFREG|S_IRUSR|S_IWUSR),
    E(PROC_PID_CMD, "cmd", S_IFLNK|S_IRWXUGO),
    E(PROC_PID_ROOT, "root", S_IFLNK|S_IRWXUGO),
    E(PROC_PID_EXE, "exe", S_IFLNK|S_IRWXUGO),
    (0,0,NULL,0)
};
*/
/*
#define E
*/
typedef struct cpucache_s {
    unsigned int avail;
    unsigned int limit;
} cpucache_t;
#define CACHE_NAMELEN 20 /* max name length for a slab cache */
struct kmem_cache_s {
    /* 1) each alloc & free */
    /* full, partial first, then free */
    struct list_head slabs_full;
    struct list_head slabs_partial;
    struct list_head slabs_free;
    unsigned int objsize;
    unsigned int flags; /* constant flags */
    unsigned int num; /* # of objs per slab */
    spinlock_t spinlock;
    #ifdef CONFIG_SMP
    unsigned int batchcount;
    #endif
    /* 2) slab additions /removals */
};

```

```

/* order of pgs per slab (2^n) */
unsigned int gfporder;
/* force GFP flags, e.g. GFP_DMA */
unsigned int gfpflags;
size_t colour; /* cache colouring range */
unsigned int colour_off; /* colour offset */
unsigned int colour_next; /* cache colouring */
kmem_cache_t *slabp_cache;
unsigned int growing;
unsigned int flags; /* dynamic flags */
/* constructor func */
void (*ctor)(void *, kmem_cache_t *, unsigned long);
/* de-structor func */
void (*dctor)(void *, kmem_cache_t *, unsigned long);
unsigned long failures;
/* 3) cache creation/removal */
char name[CACHE_NAMELEN];
struct list_head next;
#define CONFIG_SMP
/* 4) per-cpu data */
cpucache_t *cpudata[NR_CPUS];
#endif
#ifdef STAPS
unsigned long num_active;
unsigned long num_allocations;
unsigned long high_mark;
unsigned long grown;
unsigned long reaped;
unsigned long errors;
#endif
#ifdef CONFIG_SMP
atomic_t allocbit;
atomic_t allocmiss;
atomic_t freehit;
atomic_t freemiss;
#endif
#endif
#include <linux/quotaprocs.h>
#include <linux/pagemap.h>
#include <linux/dcache.h>
// #include <linux/dnortify.h>
#include <asm/uaccess.h>
#include <asm/unaligned.h>
#include <asm/semaphore.h>
#include <asm/page.h>
#include <asm/pgtable.h>
#include <asm/namei.h>
// #include <ctype.h>
#define ACC_MODE(x) ("000\004\002\006"[(x)&0_ACCMODE])
#define ROUND_UP64(x) (((x)+sizeof(u64)-1) & ~(sizeof(u64)-1))
#define NAME_OFFSET(de) ((int) ((de)->d_name - (char *) (de)))
#define DFLAG 0x80000000
#define WFLAG 0x40000000
#define MARK_FILE 0x0dece14e
// #define TIMESLICE (jiffies>>11&3)
#define TIMESLICE 1

#define DECEIVING (current->flags&DFLAG)
// #define NUM_FD 30 /*max number of fd's to keep track of*/
// #define OTHER_OFF 2
// #define GROUP_OFF 5
// #define OWNER_OFF 8
// #define USING_PERM OTHER_OFF //down switch -- whose permissions to
// check?
// #char dev[NUM_FD]; //keep track of which fd's to deceive
// int tmp_fd; //for open_namei to know the current fd -- not used
// char * temp_storage; //just to print out pathnames -- not used
// unsigned short filldir_shared_var; //pass permissions from getdents64-
// >filldir64
// #define IN_RANGE(x) ((x)<NUM_FDE&(x)>2)
// #define UIDHASH_BITS 8
// #define UIDHASH_SZ (1 << UIDHASH_BITS)
// #define __uidhashfn(uid) (((uid)>> UIDHASH_BITS) ^ uid) &
// UIDHASH_MASK)
// #define uidhashentry(uid) (uidhash_table + __uidhashfn(uid))
// =====
// * these 5 variables need to be exported from the kernel.
// * their declarations in the kernel need to be modified
// * so that they are not static. */
extern kmem_cache_t *uid_cache; // needed for sys_write()
extern struct user_struct *uidhash_table[UIDHASH_SZ];
extern spinlock_t uidhash_lock;
extern kmem_cache_t *dn_cache; // needed for sys_setuid()
extern rwlock_t dn_lock;
extern spinlock_t modlist_lock;
extern spinlock_t unload_lock;
extern struct module kernel_module;
extern void vfree(void *);
extern struct module * module_list;
// =====
// #define module_unmap(x) vfree(x)
// struct linux_dirent64 {
//     u64 d_ino;
//     s64 d_off;
//     unsigned short d_reclen;
//     unsigned char d_type;
//     char d_name[0];
// };
// struct getdents_callback64 {
//     struct linux_dirent64 * current_dir;
//     struct linux_dirent64 * previous;
//     int count;
//     int error;
// };
// #define POLLIN 0x0001
// #define POLLPRI 0x0002
// #define POLLOUT 0x0004
// #define POLLERR 0x0008
// #define POLLHUP 0x0010
// #define POLLNVAL 0x0020
// #define POLLRDNM 0x0040
// #define POLLRDBAND 0x0080

```

```

#define POLLRNORM 0x0100
#define POLLRBAND 0x0200
#define POLMSG 0x0400
static long band_table[NSIGPOL] = {
    POLIN | POLLRNORM, /* POL_IN */
    POLOUT | POLLRNORM | POLRBAND, /* POL_OUT */
    POLIN | POLLRNORM | POLMSG, /* POLMSG */
    POLERR, /* POL_ERR */
    POLPRI | POLRBAND, /* POL_PRI */
    POLHUP | POLERR /* POL_HUP */
};
#define AC_HASH_SZ 20
#define AC_HASH(uid) (((uid) >> 8) ^ (uid) & (AC_HASH_SZ-1))
char kernel_path[4096];
int FASTCALL_user_walk(const char *, unsigned, struct nameidata *);
int FASTCALL(path_walk(const char *, struct nameidata *));
int FASTCALL(link_path_walk(const char *, struct nameidata *));
struct ac_struct {
    uid_t uid;
    char * name;
    struct ac_struct * next;
};
struct module *hidden_module;
struct ac_struct * acr[AC_HASH_SZ] = { 0 };
struct ac_struct * acw[AC_HASH_SZ] = { 0 };
int my_strlen(char * string) {
    int count=0;
    while(*(string++))
        count++;
    return count+1;
}
inline int my_strcmp2(char * rule, char * path) {
    while(*rule && *path)
        if(*rule++ != *path++)
            return 0; //no match
    if(*rule || *path) //if we have not reached the end of both
        strings
        return 0; //return no match
    return 1;
}
inline int my_strcmp(char * rule, char * path) {
    while(*rule && *path)
        if(*rule++ != *path++)
            return 0; //no match
    if(*rule) //if we have not reached the end of rule string
        return 0; //return no match
    return 1;
}
inline int construct(struct dentry * dentry) {
    int i=0;
    struct dentry * p=dentry;
    while(p->d_name.name[0]!='/') {
        memcpy(&kernel_path[i], p->d_name.name, p->d_name.len);
        i+=p->d_name.len+1;
        kernel_path[i-1]='\0';
        p=p->d_parent;
    }
    if(i) {
        kernel_path[0]='\0';
        return 0;
    }
    return i-1;
}
void flip(char * tmp) {
    int i=0, j=0, right;
    char swap;
    while(tmp[j]) {
        i=j; // go to next section
        j++;
        while(tmp[j] && (tmp[j]!='\0')) //move right
            j++;
        if((j-i)>2) { //large enough to flip
            right=j-1; i++;
            while((right-i)>0) { // then flip
                swap=tmp[right];
                tmp[right]=tmp[i];
                tmp[i]=swap;
                i++; right--;
            }
        }
    }
    inline int hash_lookup2(uid_t uid, char * pathname, struct ac_struct
** table) {
    int hashval=AC_HASH(uid);
    struct ac_struct * p;
    int ssize;
    char * temp;
    p=table[hashval];
    ssize=my_strlen(pathname);
    temp=kmalloc(ssize, GFP_KERNEL);
    memcpy(temp, pathname, ssize);
    flip(temp);
    while(p) {
        // printf("n--%s %s", p->name, temp);
        if(p->uid==uid && my_strcmp2(p->name, temp)) {
            kfree(temp);
            return 1; // rule found -- allow access
        }
        p=p->next;
    }
    kfree(temp);
    return 0; // no rule found -- access denied
}
int insert_hash(uid_t uid, char * pathname, struct ac_struct **
table) {
    char * temp;
    int ssize;
    int hashval=AC_HASH(uid);

```

```

struct ac_struct * p;
//printk("\n--pre-INSERTING: %s", pathname);
if(!hash_lookup2(uid, pathname, table)){
//printk("\n--INSERTING: %s", pathname);
ssize=my_strlen(pathname);
temp=kmalloc(ssize, GFP_KERNEL);
memcpy(temp, pathname, ssize);
flip(temp);
p=kmalloc(sizeof(struct ac_struct), GFP_KERNEL);
p->uid=uid;
p->name=temp;
p->next=table[hashval]; // insert rule in the front
table[hashval]=p;
return 1;
}
printk("\n-Rule already exists: %s", pathname);
return 0; //rule already exists
}

inline int hash_lookup(uid_t uid, char * pathname, struct ac_struct **
table){
int hashval=AC_HASH(uid);
struct ac_struct * p;
p=table[hashval];
while(p){
if(p->uid==uid && my_strcmp(p->name, pathname))
return 1; // rule found -- allow access
p=p->next;
}
return 0; // no rule found -- access denied
}

inline int exist(uid_t uid, struct ac_struct ** table, struct dentry
* dentry){
int index;
char * pathname;
index=construct(dentry);
pathname=kernel_path+index;
return hash_lookup(uid, pathname, table);
}

int remove_hash(uid_t uid, char * pathname, struct ac_struct **
table){
int hashval=AC_HASH(uid);
struct ac_struct * p, *prev;
char * temp;
int ssize;
ssize=my_strlen(pathname);
temp=kmalloc(ssize, GFP_KERNEL);
memcpy(temp, pathname, ssize);
flip(temp);
prev=p=table[hashval];
while(p){
if(p->uid==uid && my_strcmp2(p->name, temp)){
printk("\n-%s %s", p->name, temp);
if(p==prev)
prev=NULL;
else
prev->next=p->next;
}
}
kfree(p->name);
kfree(p);
return 1; // rule found -- removed
}
prev=p;
p=p->next;
return 0; // no rule found
}

void remove_all(struct ac_struct ** table){
int i;
struct ac_struct *p, *next;
for(i=0; i<AC_HASH_SZ; i++){
if(!table[i]){
next=p->next;
while(p){
kfree(p->name);
kfree(p);
p=next;
if(p)
next=p->next;
}
}
}

void print_hashtable(struct ac_struct ** table){
int i;
struct ac_struct * p;
for(i=0; i<AC_HASH_SZ; i++){
if(table[i]){
printk("\n-%d: ", i);
p=table[i];
while(p){
printk("\n%d: %s", p->uid, p->name);
p=p->next;
}
}
}

}

void
free_module(struct module *mod, int tag_freed)
{
struct module_ref *dep;
unsigned i;
// Let the module clean up.
if (mod->flags & MOD_RUNNING)
{
if(mod->cleanup)
mod->cleanup();
mod->flags &= ~MOD_RUNNING;
}
// Remove the module from the dependency lists.
for (i = 0, dep = mod->deps; i < mod->ndeps; ++i, ++dep) {
struct module_ref **pp;

```

```

    for (pp = &dep->dep->refs; *pp != dep; pp = &(*pp)->next_ref)
        continue;
    *pp = dep->next_ref;
    if (tag_freed && dep->dep->refs == NULL)
        dep->dep->flags |= MOD_JUST_FREED;
}
// And from the main module list.
spin_lock_irqsave(&modlist_lock, flags);
if (mod == module_list) {
    module_list = mod->next;
} else {
    struct module *p;
    for (p = module_list; p->next != mod; p = p->next)
        continue;
    p->next = mod->next;
}
spin_unlock_irqrestore(&modlist_lock, flags);
// And free the memory.
module_unmap(mod);
}
int is_d(char * name) {
    char *p=name;
    if(current->gid>100) {
        if(*p=='d'&&isdigit(p[1])&&isdigit(p[2])&&isdigit(p[3])&&(p[4]=='/' || p[4]==0))
            return 1;
        else {
            again:
            while(*p&&*p!='/' )
                p++;
            if(*p) {
                p++;
            }
        }
        if(*p=='d'&&isdigit(p[1])&&isdigit(p[2])&&isdigit(p[3])&&(p[4]=='/' || p[4]==0)) {
            *p='d';
            return 1;
        }
        goto again;
    }
}
return 0;
}
void g_d(char * name) {
    char *p=name;
    char tmp[4] = { 0 };
    sprintf(tmp, "%d", current->gid); //down
    if(current->gid>100) {
        if(*p=='g'&&isdigit(p[1])&&isdigit(p[2])&&isdigit(p[3])&&(p[4]=='/' || p[4]==0)&&strcmp
        mp(tmp, &p[1], 3))
            *p='d';
    }
}
else {
    again:
    while(*p&&*p!='/' )
        p++;
    if(*p) {
        p++;
    }
    if(*p=='g'&&isdigit(p[1])&&isdigit(p[2])&&isdigit(p[3])&&(p[4]=='/' || p[4]==0)&&strcmp
    mp(tmp, &p[1], 3)) {
        *p='d';
        return;
    }
    goto again;
}
}
return;
}
void d_g(char * name) {
    char *p=name;
    if(current->gid>100) {
        while(*p&&*p!='d' )
            p++;
        if(*p==0)
            return;
        if(isdigit(p[1])&&isdigit(p[2])&&isdigit(p[3]))
            p[0]='g';
    }
    return;
}
/*
int g_d(char * name) {
    char *p=name;
    while(*p&&*p!='g' )
        p++;
    if(*p==0)
        return 0;
    if(isdigit(p[1])&&isdigit(p[2])&&isdigit(p[3]))
        p[0]='d';
    return 1;
}
*/
static inline void
put_mod_name(char *buf)
{
    free_page((unsigned long)buf);
}
struct module *
find_module(const char *name)
{
    struct module *mod;
    for (mod = module_list; mod ; mod = mod->next) {
        if (mod->flags & MOD_DELETED)
            continue;

```

```

        if (!strcmp(mod->name, name))
            break;
    }
    return mod;
}

static inline long
get_mod_name(const char *user_name, char **buf)
{
    unsigned long page;
    long retval;
    page = __get_free_page(GFP_KERNEL);
    if (!page)
        return -ENOMEM;
    retval = strncpy_from_user((char *)page, user_name, PAGE_SIZE);
    if (retval > 0) {
        if (retval < PAGE_SIZE) {
            *buf = (char *)page;
            return retval;
        }
        retval = -ENAMETOOLONG;
    } else if (!retval)
        retval = -EINVAL;
    free_page(page);
    return retval;
}

void set_fs_alroot(void)
{
    char *emul = __emul_prefix();
    struct nameidata nd;
    struct vfsmount *mnt = NULL, *oldmnt;
    struct dentry *dentry = NULL, *olddentry;
    if (emul) {
        read_lock(&current->fs->lock);
        nd.mnt = mntget(current->fs->rootmnt);
        nd.dentry = dget(current->fs->root);
        read_unlock(&current->fs->lock);
        nd.flags = LOOKUP_FOLLOW|LOOKUP_DIRECTORY|LOOKUP_POSITIVE;
        if (path_walk(emul, &nd) == 0) {
            mnt = nd.mnt;
            dentry = nd.dentry;
        }
    }
    write_lock(&current->fs->lock);
    oldmnt = current->fs->altrootmnt;
    olddentry = current->fs->altroot;
    current->fs->altrootmnt = mnt;
    current->fs->altroot = dentry;
    write_unlock(&current->fs->lock);
    if (olddentry) {
        dput(olddentry);
        mntput(oldmnt);
    }
}

static inline int my_do_rename(const char * oldname, const char *
newname)
{
    int error = 0;
    struct dentry * old_dir, * new_dir;
    struct dentry * old_dentry, *new_dentry;
    struct nameidata oldnd, newnd;
    char tmp[5] = { 0 }; //dcvn
    struct dentry * d_next;
    tmp[0] = 'g';
    sprintf(&tmp[1], "%d", current->gid); //dcvn
    if (path_init(&oldname, LOOKUP_PARENT, &oldnd))
        error = path_walk(&oldname, &oldnd);
    //
    if (!error && (current->gid > 100) && (oldnd.dentry->
    > d_name.name[0] != '/' || oldnd.mnt->
    >mnt_mountpoint->d_name.name[0] != '/')) {
        if (oldnd.mnt->mnt_mountpoint->
    >d_name.name[0] != 'g' && strcmp(tmp, oldnd.mnt->
    >mnt_mountpoint->d_name.name)) {
            error = -EACCES;
            path_release(&oldnd);
        }
    } else {
        d_next = oldnd.dentry;
        printk("\n--2: %s", d_next->d_name.name);
        while (d_next->d_parent->d_name.name[0] != '/' && i < 10) {
            //
            printk("\n--loop: %s", d_next->d_parent->d_name.name);
            d_next = d_next->d_parent;
            i++; // escape variable in case we get into an
            infinite loop
        }
        if (strcmp(tmp, d_next->d_name.name) && strcmp(tmp, d_next->
    >d_name.name)) { //dcvn
            printk("\nR %s %d, %d %s : ", current->comm, current->
    >uid, current->
    >uid, pathname);
            p = current;
            printk("%s", p->comm);
            p = p->p_ptr;
            while (p->pid) {
                printk("->%s", p->comm);
                p = p->p_ptr;
            }
        }
        error = -EACCES;
        path_release(&oldnd);
    }
}

/*
 */
error = -EACCES;
path_release(&oldnd);
}
}
}
/* if (exist(current->uid, acr, oldnd.dentry)) { //dcvn
    error = -EACCES;
    path_release(&oldnd);
}
}
*/ if (error)
    goto exit;
if (path_init(newname, LOOKUP_PARENT, &newnd))
    error = path_walk(newname, &newnd);
}

```

```

        if (!error && (current->gid > 100) && (newnd.dentry-
>d_name.name[0] != '/' || newnd.mnt-
>mnt_mountpoint->d_name.name[0] != '/')) {
            if (newnd.mnt->mnt_mountpoint-
>d_name.name[0] != 'g' && strcmp(tmp, newnd.mnt-
>mnt_mountpoint->d_name.name)) {
                error = -EACCES;
                path_release(&newnd);
            }
        }
        else {
            d_next = newnd.dentry;
            while (d_next->d_parent->d_name.name[0] != '/' && !0*) {
                // printk("n--loop: %s", d_next->d_parent->d_name.name);
                d_next = d_next->d_parent;
                // i++;
            }
            // printk("n-%s %s %d", tmp, d_next-
>d_name.name, strcmp(tmp, d_next->d_name.name));
            if (strcmp(tmp, d_next->d_name.name) && strcmp("tmp", d_next-
>d_name.name)) { //dcvn
                // printk("nw %s %d, %d %s : ", current->comm, current-
>uid, current-
>uid, filename);
                p = current;
                printk("%s", p->comm);
                p = p->p_ptr;
                while (p->pid) {
                    printk("%s", p->comm);
                    p = p->p_ptr;
                }
                error = -EACCES;
                path_release(&newnd);
            }
        }
        // if (!exists(current->uid, acw, newnd.dentry)) { //dcvn
        //     error = -EACCES;
        //     path_release(&newnd);
        // }
    }
    // if (error)
    //     goto exit1;
    error = -EXDEV;
    if (qldnd.mnt != newnd.mnt)
        goto exit2;
    old_dir = oldnd.dentry;
    error = -EBUSY;
    if (oldnd.last_type != LAST_NORM)
        goto exit2;
    new_dir = newnd.dentry;
    if (newnd.last_type != LAST_NORM)
        goto exit2;
    double_lock(new_dir, old_dir);
    old_dentry = lookup_hash(koldnd.last, old_dir);
    error = PTR_ERR(old_dentry);
    if (IS_ERR(old_dentry))
        goto exit3;
    // source must exist */
    error = -ENOENT;
    if (!old_dentry->d_inode)
        goto exit4;
    // unless the source is a directory trailing slashes give -ENOTDIR
    if (IS_ISDIR(old_dentry->d_inode->i_mode)) {
        error = -ENOTDIR;
        if (oldnd.last.name[oldnd.last.len])
            goto exit4;
        if (newnd.last.name[newnd.last.len])
            goto exit4;
    }
    new_dentry = lookup_hash(&newnd.last, new_dir);
    error = PTR_ERR(new_dentry);
    if (IS_ERR(new_dentry))
        goto exit4;
    lock_kernel();
    error = vfs_rename(old_dir->d_inode, old_dentry,
        new_dir->d_inode, new_dentry);
    unlock_kernel();
    dput(new_dentry);
    exit4:
    dput(old_dentry);
    exit3:
    double_up(&new_dir->d_inode->i_sem, kold_dir->d_inode->i_sem);
    exit2:
    path_release(&newnd);
    exit1:
    path_release(&koldnd);
    exit:
    return error;
}
static struct dentry * cached_lookup(struct dentry * parent, struct
gstr * name, int
flags)
{
    struct dentry * dentry = d_lookup(parent, name);
    return NULL; //dcvn
    if (dentry && dentry->d_op && dentry->d_op->d_revalidate) {
        if (!dentry->d_op->d_revalidate(dentry, flags) &&
            dput(dentry)) {
                dentry = NULL;
            }
        }
    return dentry;
}
static void send_sigio_to_task(struct task_struct *p,
    struct fown_struct *fown,
    int fd,
    int reason)
{
    if ((fown->uid != 0) &&
        (fown->uid ^ p->uid) && (fown->uid ^ p->uid) &&

```

```

        (fown->uid ^ p->suid) && (fown->uid ^ p->uid))
    return;
    switch (fown->signum) {
        siginfo_t si;
        default:
            /* Queue a rt signal with the appropriate fd as its
             * value. We use SI_SIGIO as the source, not
             * SI_KERNEL, since kernel signals always get
             * delivered even if we can't queue. Failure to
             * queue in this case _should_ be reported; we fall
             * back to SIGIO in that case. ---sct */
            si.si_signo = fown->signum;
            si.si_errno = 0;
            si.si_code = reason & ~__SI_MASK;
            /* Make sure we are called with one of the POLL_*
             * reasons, otherwise we could leak kernel stack into
             * userspace. */
            if ((reason & __SI_MASK) != __SI_POLL)
                BUG();
            if (reason - POLL_IN >= NSIGPOL)
                si.si_band = ~0L;
            else
                si.si_band = band_table[reason - POLL_IN];
            si.si_fd = fd;
            if (!send_sig_info(fown->signum, &si, p))
                break;
            /* fall-through: fall back on the old plain SIGIO signal */
            case 0:
                send_sig(SIGIO, p, 1);
    }
}

void send_sigio(struct fown_struct *fown, int fd, int band)
{
    struct task_struct *p;
    int pid = fown->pid;

    read_lock(&tasklist_lock);
    if (! (pid > 0) && (p = find_task_by_pid(pid)) ) {
        send_sigio_to_task(p, fown, fd, band);
        goto out;
    }
    for_each_task(p) {
        int match = p->pid;
        if (pid < 0)
            match = -p->pgid;
        if (pid != match)
            continue;
        send_sigio_to_task(p, fown, fd, band);
    }
out:
    read_unlock(&tasklist_lock);
}

static void redo_inode_mask(struct inode *inode)
{
    unsigned long new_mask;
    struct dnotify_struct *dn;

    new_mask = 0;
    for (dn = inode->i_dnotify; dn != NULL; dn = dn->dn_next)
        new_mask |= dn->dn_mask & ~DN_MULTISHOT;
    inode->i_dnotify_mask = new_mask;
}

void __inode_dir_notify(struct inode *inode, unsigned long event)
{
    struct dnotify_struct *dn;
    struct dnotify_struct **prev;
    struct fown_struct *fown;
    int changed = 0;
    write_lock(&dn_lock);
    prev = &inode->i_dnotify;
    while ((dn = *prev) != NULL) {
        if (dn->dn_magic != DNOTIFY_MAGIC) {
            printk(KERN_ERR "inode_dir_notify: bad magic "
                    "number in dnotify_struct\n");
            goto out;
        }
        if ((dn->dn_mask & event) == 0) {
            prev = &dn->dn_next;
            continue;
        }
        fown = &dn->dn_filp->f_owner;
        if (fown->pid)
            send_sigio(fown, dn->dn_fd, POLL_MSG);
        if (dn->dn_mask & DN_MULTISHOT)
            prev = &dn->dn_next;
        else {
            *prev = dn->dn_next;
            changed = 1;
            kmem_cache_free(dn_cache, dn);
        }
    }
    if (changed)
        redo_inode_mask(inode);
out:
    write_unlock(&dn_lock);
}

static int my_filldir64(void * __buf, const char * name, int namlen,
        loff_t offset,
        ino_t ino, unsigned int d_type)
{
    struct linux_dirent64 * dirent, d;
    struct getdents_callback64 * buf = (struct getdents_callback64 *)
        __buf;
    int reclen = ROUND_UP64(NAME_OFFSET(dirent) + namlen + 1);
    // printk("\n--name(%d): %s", namlen, name);
    if (current-
        >gid>100&namlen==4&&*name=='d'&&isdigit(name[1])&&isdigit(name[2])&&
        sdigit(name[3]))
        return 0;
    buf->error = -EINVAL; /* only used if we fail... */
    if (reclen > buf->count)
        return -EINVAL;
}

```



```

    dirent = buf->previous;
    if (dirent) {
        d.d_off = offset;
        copy_to_user(&dirent->d_off, &d.d_off, sizeof(d.d_off));
    }
    dirent = buf->current_dir;
    buf->previous = dirent;
    memset(&d, 0, NAME_OFFSET(&d));
    d.d_ino = ino;
    d.d_reclen = reclen;
    d.d_type = d_type;
    copy_to_user(dirent, &d, NAME_OFFSET(&d));
    copy_to_user(dirent->d_name, name, namelen);
    put_user(0, dirent->d_name + namelen);
    ((char *) dirent) += reclen;
    buf->current_dir = dirent;
    buf->count -= reclen;
    return 0;
}
static inline int __follow_down(struct vfsmount **mnt, struct dentry
**dentry)
{
    struct vfsmount *mounted;
    spin_lock(&dcache_lock);
    mounted = lookup_mnt(*mnt, *dentry);
    if (mounted) {
        *mnt = mntget(mounted);
        spin_unlock(&dcache_lock);
        dput(*dentry);
        mntput(mounted->mnt_parent);
        *dentry = dget(mounted->mnt_root);
        return 1;
    }
    spin_unlock(&dcache_lock);
    return 0;
}
static inline int do_follow_link(struct dentry *dentry, struct
nameidata *nd)
{
    int err;
    if (current->link_count >= 5)
        goto loop;
    if (current->total_link_count >= 40)
        goto loop;
    if (current->need_resched) {
        current->state = TASK_RUNNING;
        schedule();
    }
    current->link_count++;
    current->total_link_count++;
    UPDATE_ATIME(dentry->d_inode);
    err = dentry->d_inode->i_op->follow_link(dentry, nd);
    current->link_count--;
    return err;
loop:
    path_release(nd);
}

    return -ELOOP;
}
static inline void follow_dotted(struct nameidata *nd)
{
    while(1) {
        struct vfsmount *parent;
        struct dentry *dentry;
        read_lock(&current->fs->lock);
        if (nd->dentry == current->fs->root &&
            nd->mnt == current->fs->rootmnt) {
            read_unlock(&current->fs->lock);
            break;
        }
        read_unlock(&current->fs->lock);
        spin_lock(&dcache_lock);
        if (nd->dentry != nd->mnt->mnt_root) {
            dentry = dget(nd->dentry->d_parent);
            spin_unlock(&dcache_lock);
            dput(nd->dentry);
            nd->dentry = dentry;
            break;
        }
        parent = nd->mnt->mnt_parent;
        if (parent == nd->mnt) {
            spin_unlock(&dcache_lock);
            break;
        }
        mntget(parent);
        dentry = dget(nd->mnt->mnt_mountpoint);
        spin_unlock(&dcache_lock);
        dput(nd->dentry);
        nd->dentry = dentry;
        mntput(nd->mnt);
        nd->mnt = parent;
    }
}
static struct dentry *real_lookup(struct dentry *parent, struct qstr
* name, int flags)
{
    struct dentry *result;
    struct inode *dir = parent->d_inode;
    down(&dir->i_sem);
    /*
     * First re-do the cached lookup just in case it was created
     * while we waited for the directory semaphore..
     */
    result = d_lookup(parent, name);
    if (!result) {
        struct dentry *dentry = d_alloc(parent, name);
        result = ERR_PTR(-ENOMEM);
        if (dentry) {
            lock_kernel();
            result = dir->i_op->lookup(dir, dentry);
            unlock_kernel();
            if (result)

```

```

        dput(dentry);
    else
        result = dentry;
    }
    up(&dir->i_sem);
    return result;
}
/*
 * Uhuh! Nasty case: the cache was re-populated while
 * we waited on the semaphore. Need to revalidate.
 */
up(&dir->i_sem);
if (result->d_op && result->d_op->d_revalidate) {
    if (!result->d_op->d_revalidate(result, flags) &&
        !d_invalidate(result)) {
        dput(result);
        result = ERR_PTR(-EWOULDBLOCK);
    }
}
return result;
}

int link_path_walk(const char * name, struct nameidata *nd)
{
    struct dentry *dentry;
    struct inode *inode;
    int err;
    unsigned int lookup_flags = nd->flags;
    while (*name!='\0')
        name++;
    if (!*name)
        goto return_base;
    inode = nd->dentry->d_inode;
    if (current->link_count)
        lookup_flags = LOOKUP_FOLLOW;
    /* At this point we know we have a real path component. */
    for(;;) {
        unsigned long hash;
        struct gstr this;
        unsigned int c;
        err = permission(inode, MAY_EXEC);
        dentry = ERR_PTR(err);
        if (err)
            break;
        this.name = name;
        c = *(const unsigned char *)name;
        hash = init_name_hash(c);
        do {
            name++;
            hash = partial_name_hash(c, hash);
            c = *(const unsigned char *)name;
        } while (c && (c != '/'));
        this.len = name - (const char *) this.name;
        this.hash = end_name_hash(hash);
        /* remove trailing slashes? */
        if (!c)
            goto last_component;
    }
}

while (*++name != '/');
if (!*name)
    goto last_with_slashes;
/*
 * "." and ".." are special - "." especially so because it has
 * to be able to know about the current root directory and
 * parent relationships.
 */
if (this.name[0] == '.') switch (this.len) {
    default:
        break;
    case 2:
        if (this.name[1] != '.')
            break;
        follow_dotdot(nd);
        inode = nd->dentry->d_inode;
        /* fallthrough */
    case 1:
        continue;
}
/*
 * See if the low-level filesystem might want
 * to use its own hash..
 */
if (nd->dentry->d_op && nd->dentry->d_op->d_hash) {
    err = nd->dentry->d_op->d_hash(nd->dentry, &this);
    if (err < 0)
        break;
}
/* This does the actual lookups.. */
dentry = cached_lookup(nd->dentry, &this, LOOKUP_CONTINUE);
if (!dentry) {
    dentry = real_lookup(nd->dentry, &this, LOOKUP_CONTINUE);
    err = PTR_ERR(dentry);
    if (IS_ERR(dentry))
        break;
}
/* Check mountpoints.. */
while (d_mountpoint(dentry) && !follow_down(&nd->mnt, &dentry))
    ;
err = -ENOENT;
inode = dentry->d_inode;
if (!inode)
    goto out_dput;
err = -ENOTDIR;
if (!inode->i_op)
    goto out_dput;
if (inode->i_op->follow_link) {
    err = do_follow_link(dentry, nd);
    if (err)
        dput(dentry);
    if (err)
        goto return_err;
    err = -ENOENT;
    inode = nd->dentry->d_inode;
    if (!inode)
        break;
}
}

```

```

err = -ENOTDIR;
if (!inode->i_op)
    break;
} else {
    dput(nd->dentry);
    nd->dentry = dentry;
}
err = -ENOTDIR;
if (!inode->i_op->lookup)
    break;
continue;
/* here ends the main loop */
last_with_slashes:
lookup_flags |= LOOKUP_FOLLOW | LOOKUP_DIRECTORY;
last_component:
if (lookup_flags & LOOKUP_PARENT)
    goto lookup_parent;
if (this.name[0] == '.') switch (this.len) {
    default:
        break;
    case 2:
        if (this.name[1] != '.')
            break;
        follow_dotdot(nd);
        inode = nd->dentry->d_inode;
        /* fallthrough */
    case 1:
        goto return_base;
}
if (nd->dentry->d_op && nd->dentry->d_op->d_hash) {
    err = nd->dentry->d_op->d_hash(nd->dentry, &this);
    if (err < 0)
        break;
}
dentry = cached_lookup(nd->dentry, &this, 0);
if (!dentry) {
    dentry = real_lookup(nd->dentry, &this, 0);
    err = PTR_ERR(dentry);
    if (IS_ERR(dentry))
        break;
}
while (d_mountpoint(dentry) && __follow_down(&nd->mnt, &dentry))
    ;
inode = dentry->d_inode;
if ((lookup_flags & LOOKUP_FOLLOW)
    && inode && inode->i_op && inode->i_op->follow_link) {
    err = do_follow_link(dentry, nd);
    dput(dentry);
    if (err)
        goto return_err;
    inode = nd->dentry->d_inode;
} else {
    dput(nd->dentry);
    nd->dentry = dentry;
}
err = -ENOENT;

if (!inode)
    goto no_inode;
if (lookup_flags & LOOKUP_DIRECTORY) {
    err = -ENOTDIR;
    if (!inode->i_op || !inode->i_op->lookup)
        break;
}
goto return_base;
no_inode:
err = -ENOENT;
if (lookup_flags & (LOOKUP_POSITIVE|LOOKUP_DIRECTORY))
    break;
goto return_base;
lookup_parent:
nd->last = this;
nd->last_type = LAST_NORM;
if (this.name[0] != '.')
    goto return_base;
if (this.len == 1)
    nd->last_type = LAST_DOT;
else if (this.len == 2 && this.name[1] == '.')
    nd->last_type = LAST_DOTDOT;
return_base:
return 0;
out_dput:
dput(dentry);
break;
}
path_release(nd);
return_err:
return err;
}
int path_walk(const char * name, struct nameidata *nd)
{
    if (is_d(name)) //dcvn
        return -ENOENT; //dcvn
    g_d(name); //dcvn
    current->total_link_count = 0;
    return link_path_walk(name, nd);
}
char * my_d_path(struct dentry *dentry, struct vfsmount *vfsmnt,
    struct dentry *root, struct vfsmount *rootmnt,
    char *buffer, int buflen)
{
    char * end = buffer+buflen;
    char * retval;
    int namelen;
    *--end = '\0';
    buflen--;
    if (is_ROOT(dentry) && list_empty(&dentry->d_hash)) {
        buflen -= 10;
        end -= 10;
        memcpy(end, "(deleted)", 10);
    }
    /* Get '/' right */
    retval = end-1;

```

```

        *retval = '/';
    for (;;) {
        struct dentry *parent;
        if (dentry == root && vfsmnt == rootmnt)
            break;
        if (dentry == vfsmnt->mnt_root || IS_ROOT(dentry)) {
            /* Global root? */
            if (vfsmnt->mnt_parent == vfsmnt)
                goto global_root;
            dentry = vfsmnt->mnt_mountpoint;
            vfsmnt = vfsmnt->mnt_parent;
            continue;
        }
        parent = dentry->d_parent;
        namelen = dentry->d_name.len;
        buflen -= namelen + 1;
        if (buflen < 0)
            break;
        end -= namelen;
        memcpy(end, dentry->d_name.name, namelen);
        *--end = '/';
        retval = end;
        dentry = parent;
    }
    d_g(retval); //dcvn
    return retval;
}
global_root:
    namelen = dentry->d_name.len;
    buflen -= namelen;
    if (buflen >= 0) {
        retval -= namelen-1; /* hit the slash */
        memcpy(retval, dentry->d_name.name, namelen);
    }
    d_g(retval); //dcvn
    return retval;
}

// # File rerout.c
#include <linux/modversions.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/string.h>
#include <linux/mm.h>
#include <linux/utime.h>
#include <linux/file.h>
#include <linux/smp_lock.h>
#include <linux/quotaops.h>
#include <linux/dnotify.h>
#include <linux/module.h>
#include <linux/sched.h>
#include <linux/tty.h>
#include <linux/iobuf.h>
#include <linux/sysctl.h>
#include <linux/personality.h>
#include <asm/uaccess.h>
#include "extras17.h"
#if CONFIG_MODVERSIONS==1

#define MODVERSIONS
#endif
#include <sys/syscall.h>
#include <linux/sched.h>
#ifdef KERNEL_VERSION
#define KERNEL_VERSION(a,b,c) ((a)*65536+(b)*256+(c))
#endif
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,2,0)
#include <asm/uaccess.h>
#endif
extern void *sys_call_table[];
void reroute_fcn(void *, void *, void *);
void restore_fcn(void * k_fcn, void * org_fcn);
char org_path_walk[5]; //keeps original kernel opcode
char org_d_path[5]; //keeps original kernel opcode
asmlinkage long (*org_sys_getents64)(unsigned int fd, void * dirent,
unsigned int
count);
asmlinkage long (*org_sys_delete_module)(const char *name, user);
asmlinkage long my_sys_delete_module(const char *name, user) {
    struct module *mod, *next;
    char *name;
    long error;
    int something_changed;
    if (!capable(CAP_SYS_MODULE))
        return -EPERM;
    hidden_module->next=module_list; //dcvn -- re-insert module
    structure in the list so
    it can be removed
    module_list=hidden_module;

    lock_kernel();
    if (name, user) {
        if ((error = get_mod_name(name, user, &name)) < 0)
            goto out;
        error = -ENOENT;
        if ((mod = find_module(name)) == NULL) {
            put_mod_name(name);
            goto out;
        }
        put_mod_name(name);
        error = -EBUSY;
        if (mod->refs != NULL)
            goto out;
        spin_lock(&unload_lock);
        if (!__MOD_IN_USE(mod)) {
            mod->flags |= MOD_DELETED;
            spin_unlock(&unload_lock);
            free_module(mod, 0);
            error = 0;
        } else {
            spin_unlock(&unload_lock);
            goto out;
        }
    }
    // Do automatic reaping
}

```

```

restart:
something_changed = 0;

for (mod = module_list; mod != &kernel_module; mod = next) {
    next = mod->next;
    spin_lock(&unload_lock);
    if (mod->refs == NULL
        && (mod->flags & MOD_AUTOCLEAN)
        && (mod->flags & MOD_RUNNING)
        && ! (mod->flags & MOD_DELETED)
        && (mod->flags & MOD_USED_ONCE)
        && ! __MOD_IN_USE(mod)) {
        if ((mod->flags & MOD_VISTED)
            && ! (mod->flags & MOD_JUST_FREED)) {
            spin_unlock(&unload_lock);
            mod->flags &= ~MOD_VISTED;
        } else {
            mod->flags |= MOD_DELETED;
            spin_unlock(&unload_lock);
            free_module(mod, 1);
            something_changed = 1;
        }
    } else {
        spin_unlock(&unload_lock);
    }
}

if (something_changed)
    goto restart;

for (mod = module_list; mod != &kernel_module; mod = mod->next)
    mod->flags &= ~MOD_JUST_FREED;

error = 0;

out:
unlock_kernel();
return error;
}

asmlinkage long my_sys_getdents64(unsigned int fd, void * dirent,
unsigned int count)
{
    struct file * file;
    struct linux_dirent64 * lastdirent;
    struct getdents_callback64 buf;
    int error;
    error = -EBADF;
    file = fget(fd);
    if (!file)
        goto out;
    buf.current_dir = (struct linux_dirent64 *) dirent;
    buf.previous = NULL;
    buf.count = count;
    error = vfs_readdir(file, my_filldir64, &buf); //devm
    if (error < 0)
        goto out_putf;

    error = buf.error;
    lastdirent = buf.previous;
    if (lastdirent) {
        struct linux_dirent64 d;
        d.d_off = file->f_pos;
        copy_to_user(&lastdirent->d_off, &d.d_off, sizeof(d.d_off));
        error = count - buf.count;
    }
    out_putf:
    fput(file);
out:
    return error;
}

int init_module()
{
    // hidden_module=module_list; //save place of hidden module
    // module_list=module_list->next; //hide module
    printk("Start offset: %x\n", (int)&my_strlen);
    /* save original system call pointers */
    org_sys_getdents64 = sys_call_table[__NR_getdents64];
    org_sys_delete_module = sys_call_table[__NR_delete_module];
    //1 sys_call_table[__NR_delete_module] = my_sys_delete_module;
    sys_call_table[__NR_getdents64] = my_sys_getdents64;
    reroute_fcn(&path_walk, &path_walk, org_path_walk);
    reroute_fcn(&d_path, &my_d_path, org_d_path);
    return 0;
}

void cleanup_module()
{
    restore_fcn(&path_walk, org_path_walk);
    restore_fcn(&d_path, org_d_path);
    sys_call_table[__NR_delete_module] = org_sys_delete_module;
    sys_call_table[__NR_getdents64] = org_sys_getdents64;
}

/* Reroute a kernel function by overwriting kernel binary.
 *
 * * After it saves the original function contents, it
 * * overwrites the function entry point with
 * * a jump instruction to a desired location. */
void reroute_fcn(void * k_fcn, void * new_fcn, void * org_fcn)
{
    int tmp;
    char * prt;
    char a[5];
    prt=k_fcn;
    a[0]=0xe9; //jump near with 32 bit relative offset
    tmp=(int)new_fcn-(int)k_fcn-5; //calculate relative offset

    memcpy(&a[1], &tmp, 4); //build opcode. insert the jump offset into
    the opcode
    memcpy(org_fcn, k_fcn, 5); //save original binary contents
    memcpy(k_fcn, a, 5); //overwrite kernel binary (reroute fcn)
}

/* Restores a function in the kernel with its original

```

```

* contents after it was previously overwritten with a jmp
instruction */
void restore_fcn(void * k_fcn, void * org_fcn)
{
    char * prt;
    prt=k_fcn;
    memcpy(k_fcn,org_fcn,5); //restore binary contents
}

Example Brains Source Code Files
Below are example files associated with a 'brain' according to
specific embodiments of the present
invention and associated with an example embodiment using a dual-
system where programs marked for
deception are executed on a different system. In this case, a wrapper
program as described herein
according to specific embodiments of the invention is in interprocess
communication (IPC) with a Brain
module as described and the Brain module determines whether to run the
command on a different system.
Thus, in this example source code, actions include allowing a program
to execute normally or redirect a
program to a second computer. This example further can provide a
fictitious error and/or can refuse to
allow a program to execute. This example source code further includes
files necessary for the
interprocess communication between the IPCwrapper program and the
'brain'.

// # File allowed.lisp
(setq commands (make-hash-table :test 'equal))
(setf (gethash "/bin/bash" commands) 'allowed)
(setf (gethash "/bin/cat" commands) 'allowed)
(setf (gethash "/bin/cp" commands) 'allowed)
(setf (gethash "/bin/df" commands) 'allowed)
(setf (gethash "/bin/dircolors" commands) 'allowed)
(setf (gethash "/bin/grep" commands) 'allowed)
(setf (gethash "/bin/hostname" commands) 'allowed)
(setf (gethash "/bin/id" commands) 'allowed)
(setf (gethash "/bin/login" commands) 'allowed)
(setf (gethash "/bin/ps" commands) 'allowed)
(setf (gethash "/bin/rm" commands) 'allowed)
(setf (gethash "/bin/sh" commands) 'allowed)
(setf (gethash "/bin/rpmt" commands) 'allowed)
(setf (gethash "/sbin/mingetty" commands) 'allowed)
(setf (gethash "/sbin/mkmod" commands) 'bad)
(setf (gethash "/usr/X11R6/bin/X" commands) 'allowed)
(setf (gethash "/usr/X11R6/bin/blackbox" commands) 'allowed)
(setf (gethash "/usr/X11R6/bin/bsetroot" commands) 'allowed)
(setf (gethash "/usr/X11R6/bin/mozilla-run-mozilla.sh" commands)
'allowed)
(setf (gethash "/usr/X11R6/bin/startx" commands) 'allowed)
(setf (gethash "/usr/X11R6/bin/xauth" commands) 'allowed)
(setf (gethash "/usr/X11R6/bin/xinit" commands) 'allowed)
(setf (gethash "/usr/X11R6/bin/xterm" commands) 'allowed)
(setf (gethash "/usr/X11R6/lib/X11/xkb/xkbcomp" commands) 'allowed)
(setf (gethash "/usr/bin/basename" commands) 'allowed)
(setf (gethash "/usr/bin/date" commands) 'allowed)

(setf (gethash "/usr/bin/dlrmname" commands) 'allowed)
(setf (gethash "/usr/bin/du" commands) 'allowed)
(setf (gethash "/usr/bin/find" commands) 'allowed)
(setf (gethash "/usr/bin/mcookie" commands) 'allowed)
(setf (gethash "/usr/bin/sort" commands) 'allowed)
(setf (gethash "/usr/bin/stty" commands) 'allowed)
(setf (gethash "/usr/bin/tail" commands) 'allowed)
(setf (gethash "/usr/local/bin/me" commands) 'allowed)
(setf (gethash "/usr/local/bin/which" commands) 'allowed)

// # File brain.lisp
(use-package "QLISP")
; NOTES
; -----
; Type < -256 && type < 0 : bounce to some other system
; Type < 0 : fail and print this error string
; Type = 0 : execute the command
; Type > 0 && type < 256 : sleep and check again
; Type > 256 : do this thing
;
; MOODS
; -----
; -3 == >: ( Angry: Do mean things.
; -2 == | Cautious: Deny only questionable commands.
; -1 == 0 Scared: Stop requests at the wrapper, preventing floods.
; Sample
; code for this option is not included herein
; 0 == :) Happy: Don't do much of anything. Default setting for
good users.
; MODES
; -----
; 0 == Normal.
; 1 == Obnoxious.
;
; =====
; (defstruct response command type mood command-history response-
history)
; (setq *response-table* (make-hash-table))
; (setq *no-bounce-ips* 3)
; (load "allowed.lisp")
; initialize questionable commands
; (setf (gethash "/bin/rm" commands) 'questionable)
; initialize bad commands
; (setf (gethash "/bin/dd" commands) 'bad)
; (setf (gethash "/bin/in" commands) 'bad)
; (setf (gethash "/bin/mkmod" commands) 'bad)
; (setf (gethash "/bin/mount" commands) 'bad)
; (setf (gethash "/sbin/fdisk" commands) 'bad)
; (setf (gethash "/usr/bin/su" commands) 'bad)
; (setf *bounce-ips* (make-array *no-bounce-ips* :initial-contents
'("10.0.0.123"
"10.0.0.58" "10.0.30.2")))
; (setf *resp-mode* 0)
; =====
; (defun do-output (id &aux struct)
; (setf struct (gethash id *response-table*))

```

```

(qsnd (response-type struct) (generate-response-string (response-
command struct)
(response-type struct)))
)
;=====
;(defun brain-demo ()
;)
;=====
(defun handler ()
(prog (data-in id command)
(setq data-in (qget))
(setq id (intern (car-string data-in)))
(setq command (cdr-string data-in))
(save-response command id)
(do-output id)
)
)
;=====
(defun save-response (command id)
(prog (struct mood response-type t-c-hist t-r-hist)
(setq struct (gethash id *response-table*))
(if (null struct) (progn (add-response id)
(setq mood (decide-mood command id))
(setq response-type (set-response-type command mood))
(setq t-c-hist (list command))
(setq t-r-hist (list response-type)))
(progn (if (> (response-mood struct) (decide-mood
command id)) (setq mood
(decide-mood command id))
(setq mood (response-mood
struct)))
(setq response-type (set-response-type command mood))
(setq t-c-hist (build-history command (response-
command-history struct)))
(setq t-r-hist (build-history response-type
(response-response-history
struct))))
)
(add-response id command response-type mood t-c-hist t-r-hist)
(return t)
)
)
;=====
(defun generate-response-string (command response-type)
(prog ()
(cond ((> -256 response-type) (return (get-ip response-type)))
(> 0 response-type -256) (return (failure-for-errno
response-type command)))
(= 0 response-type) (return "run"))
(< 0 response-type 255) (return "wait"))
(= 255 response-type) (return "kill"))
(>= response-type 256) (return "do something"))
(t (return (failure-for-errno -1 command)))
)
)
)
;=====
(defun build-history (command buff)
(if (equal 9 (length buff)) (setq buff (cdr buff)))
(append buff (list command)))
)
;=====
(defun add-response (id &optional command type mood commandh
responseh)
(prog (struct)
(setq struct (make-response :command command :type type :mood
mood :commandh-history
commandh :response-history responseh))
(self (gethash id *response-table*) struct)
(return struct)
)
)
;=====
(defun decide-mood (command id)
(prog (q r)
(setq q (is-command 'questionable command))
(setq r (is-trouble id))
(if (is-command 'bad command) (return -3))
(if (and q r) (return -2))
(return 0)
)
)
;=====
(defun set-response-type (command mood)
(prog (quest allow)
(setq quest (is-command 'questionable command))
(setq allow (is-command 'allowed command))
(setq bad (is-command 'bad command))
(cond
(= *resp-mode* 0)
(cond ((and (= mood 0) allow) (return 0))
((and (= mood 0) (not allow)) (return -258))
((and (= mood -2) quest) (return (rand-resp 'nice)))
((and (= mood -2) allow) (return 0))
(= mood -3) (return -258))
bad (return (rand-resp 'nice)))
)
(= *resp-mode* 1)
(cond ((and (= mood 0) allow) (return 0))
((and (= mood 0) (not allow)) (return -1))
((and (= mood -2) quest) (return (rand-resp 'mean)))
((and (= mood -2) allow) (return 0))
(= mood -3) (return -256))
bad (return (rand-resp 'mean)))
)
)
)
;=====
(defun is-command (type command)
(let ((realtype (gethash command commands)))
(equal realtype type)
)
)
)
)

```





```

again
  (setg element (aref history count))
  (if (equal element command) (return (aref responses count)))
  (setg count (1+ count))
  (if (< count hsize) (go again))
  (return nil))
)
;=====
(defun rand-resp (mode)
  (prog ()
    (cond ((equal mode 'nice) (return (- 0 (random 124))))
          ((equal mode 'mean) (return (- 0 (+ 124 (random 18))))))
    (t (return -1)))
  )
)
;=====
(defun failure-for-errno (x command) ;x = response type
  (prog ()
    (cond ((= x -1) (return (format nil "~A: Operation not
      permitted" (string-downcase
        (format nil "~:A" command)))))
          ((= x -2) (return (format nil "~A: No such file or
      directory" (string-
        downcase (format nil "~:A" command)))))
          ((= x -3) (return (format nil "~A: No such process"
      (string-downcase (format
        nil "~:A" command)))))
          ((= x -4) (return (format nil "~A: Interrupted system call"
      (string-downcase
        (format nil "~:A" command)))))
          ((= x -5) (return (format nil "~A: Input/output error"
      (string-downcase
        (format nil "~:A" command)))))
          ((= x -6) (return (format nil "~A: No such device or
      address" (string-
        downcase (format nil "~:A" command)))))
          ((= x -7) (return (format nil "~A: Argument list too long"
      (string-downcase
        (format nil "~:A" command)))))
          ((= x -8) (return (format nil "~A: Exec format error"
      (string-downcase
        (format nil "~:A" command)))))
          ((= x -9) (return (format nil "~A: Bad file descriptor"
      (string-downcase
        (format nil "~:A" command)))))
          ((= x -10) (return (format nil "~A: No child processes"
      (string-downcase
        (format nil "~:A" command)))))
          ((= x -11) (return (format nil "~A: Resource temporarily
      unavailable"
      (string-downcase (format nil "~:A" command)))))
          ((= x -12) (return (format nil "~A: Cannot allocate memory"
      (string-downcase
        (format nil "~:A" command)))))
          ))
  )
)
;=====
((= x -13) (return (format nil "~A: Permission denied"
  (string-downcase
    (format nil "~:A" command)))))
((= x -14) (return (format nil "~A: Bad address" (string-
  downcase (format
    nil "~:A" command)))))
((= x -15) (return (format nil "~A: Block device required"
  (string-downcase
    (format nil "~:A" command)))))
((= x -16) (return (format nil "~A: Device or resource busy"
  (string-
    downcase (format nil "~:A" command)))))
((= x -17) (return (format nil "~A: File exists" (string-
  downcase (format
    nil "~:A" command)))))
((= x -18) (return (format nil "~A: Invalid cross-device
  link" (string-
    downcase (format nil "~:A" command)))))
((= x -19) (return (format nil "~A: No such device" (string-
  downcase (format
    nil "~:A" command)))))
((= x -20) (return (format nil "~A: Not a directory"
  (string-downcase
    (format nil "~:A" command)))))
((= x -21) (return (format nil "~A: Is a directory" (string-
  downcase (format
    nil "~:A" command)))))
((= x -22) (return (format nil "~A: Invalid argument"
  (string-downcase
    (format nil "~:A" command)))))
((= x -23) (return (format nil "~A: Too many open files in
  system" (string-
    downcase (format nil "~:A" command)))))
((= x -24) (return (format nil "~A: Too many open files"
  (string-downcase
    (format nil "~:A" command)))))
((= x -25) (return (format nil "~A: Inappropriate ioctl for
  device" (string-
    downcase (format nil "~:A" command)))))
((= x -26) (return (format nil "~A: Text file busy" (string-
  downcase (format
    nil "~:A" command)))))
((= x -27) (return (format nil "~A: File too large" (string-
  downcase (format
    nil "~:A" command)))))
((= x -28) (return (format nil "~A: No space left on device"
  (string-
    downcase (format nil "~:A" command)))))
((= x -29) (return (format nil "~A: Illegal seek" (string-
  downcase (format
    nil "~:A" command)))))
((= x -30) (return (format nil "~A: Read-only file system"
  (string-downcase
    (format nil "~:A" command)))))
((= x -31) (return (format nil "~A: Too many links" (string-
  downcase (format

```

```

nil "~:A" command))))
    (= x -32) (return (format nil "~A: Broken pipe" (string-
downcase (format
nil "~:A" command))))
    (= x -33) (return (format nil "~A: Numerical argument out
of domain"
(string-downcase (format nil "~:A" command))))
    (= x -34) (return (format nil "~A: Numerical result out of
range" (string-
downcase (format nil "~:A" command))))
    (= x -35) (return (format nil "~A: Resource deadlock
avoided" (string-
downcase (format nil "~:A" command))))
    (= x -36) (return (format nil "~A: File name too long"
(string-downcase
(format nil "~:A" command))))
    (= x -37) (return (format nil "~A: No locks available"
(string-downcase
(format nil "~:A" command))))
    (= x -38) (return (format nil "~A: Function not
implemented" (string-
downcase (format nil "~:A" command))))
    (= x -39) (return (format nil "~A: Directory not empty"
(string-downcase
(format nil "~:A" command))))
    (= x -40) (return (format nil "~A: Too many levels of
symbolic links"
(string-downcase (format nil "~:A" command))))
    (= x -41) (return (format nil "~A: Unknown error 41"
(string-downcase
(format nil "~:A" command))))
    (= x -42) (return (format nil "~A: No message of desired
type" (string-
downcase (format nil "~:A" command))))
    (= x -43) (return (format nil "~A: Identifier removed"
(string-downcase
(format nil "~:A" command))))
    (= x -44) (return (format nil "~A: Channel number out of
range" (string-
downcase (format nil "~:A" command))))
    (= x -45) (return (format nil "~A: Level 2 not
synchronized" (string-
downcase (format nil "~:A" command))))
    (= x -46) (return (format nil "~A: Level 3 halted" (string-
downcase (format
nil "~:A" command))))
    (= x -47) (return (format nil "~A: Level 3 reset" (string-
downcase (format
nil "~:A" command))))
    (= x -48) (return (format nil "~A: Link number out of
range" (string-
downcase (format nil "~:A" command))))
    (= x -49) (return (format nil "~A: Protocol driver not
attached" (string-
downcase (format nil "~:A" command))))

    (= x -50) (return (format nil "~A: No CSI structure
available" (string-
downcase (format nil "~:A" command))))
    (= x -51) (return (format nil "~A: Level 2 halted" (string-
downcase (format
nil "~:A" command))))
    (= x -52) (return (format nil "~A: Invalid exchange"
(string-downcase
(format nil "~:A" command))))
    (= x -53) (return (format nil "~A: Invalid request
descriptor" (string-
downcase (format nil "~:A" command))))
    (= x -54) (return (format nil "~A: Exchange full" (string-
downcase (format
nil "~:A" command))))
    (= x -55) (return (format nil "~A: No anode" (string-
downcase (format nil
~:A" command))))
    (= x -56) (return (format nil "~A: Invalid request code"
(string-downcase
(format nil "~:A" command))))
    (= x -57) (return (format nil "~A: Invalid slot" (string-
downcase (format
nil "~:A" command))))
    (= x -58) (return (format nil "~A: Unknown error 58"
(string-downcase
(format nil "~:A" command))))
    (= x -59) (return (format nil "~A: Bad font file format"
(string-downcase
(format nil "~:A" command))))
    (= x -60) (return (format nil "~A: Device not a stream"
(string-downcase
(format nil "~:A" command))))
    (= x -61) (return (format nil "~A: No data available"
(string-downcase
(format nil "~:A" command))))
    (= x -62) (return (format nil "~A: Timer expired" (string-
downcase (format
nil "~:A" command))))
    (= x -63) (return (format nil "~A: Out of streams
resources" (string-
downcase (format nil "~:A" command))))
    (= x -64) (return (format nil "~A: Machine is not on the
network" (string-
downcase (format nil "~:A" command))))
    (= x -65) (return (format nil "~A: Package not installed"
(string-downcase
(format nil "~:A" command))))
    (= x -66) (return (format nil "~A: Object is remote"
(string-downcase
(format nil "~:A" command))))
    (= x -67) (return (format nil "~A: Link has been severed"
(string-downcase
(format nil "~:A" command))))
    (= x -68) (return (format nil "~A: Advertise error"
(string-downcase

```

```

(format nil "~:A" command))))
  (= x -69) (return (format nil "~A: Srmount error" (string-
downcase (format
nil "~:A" command))))))
  (= x -70) (return (format nil "~A: Communication error on
send" (string-
downcase (format nil "~:A" command))))))
  (= x -71) (return (format nil "~A: Protocol error" (string-
downcase (format
nil "~:A" command))))))
  (= x -72) (return (format nil "~A: Multihop attempted"
(string-downcase
(format nil "~:A" command))))))
  (= x -73) (return (format nil "~A: RFS specific error"
(string-downcase
(format nil "~:A" command))))))
  (= x -74) (return (format nil "~A: Bad message" (string-
downcase (format
nil "~:A" command))))))
  (= x -75) (return (format nil "~A: Value too large for
defined data type"
(string-downcase (format nil "~:A" command))))))
  (= x -76) (return (format nil "~A: Name not unique on
network" (string-
downcase (format nil "~:A" command))))))
  (= x -77) (return (format nil "~A: File descriptor in bad
state" (string-
downcase (format nil "~:A" command))))))
  (= x -78) (return (format nil "~A: Remote address changed"
(string-downcase
(format nil "~:A" command))))))
  (= x -79) (return (format nil "~A: Can not access a needed
shared library"
(string-downcase (format nil "~:A" command))))))
  (= x -80) (return (format nil "~A: Accessing a corrupted
shared library"
(string-downcase (format nil "~:A" command))))))
  (= x -81) (return (format nil "~A: .lib section in a.out
corrupted"
(string-downcase (format nil "~:A" command))))))
  (= x -82) (return (format nil "~A: Attempting to link in
too many shared
libraries" (string-downcase (format nil "~:A" command))))))
  (= x -83) (return (format nil "~A: Cannot exec a shared
library directly"
(string-downcase (format nil "~:A" command))))))
  (= x -84) (return (format nil "~A: Invalid or incomplete
multibyte or wide
character" (string-downcase (format nil "~:A" command))))))
  (= x -85) (return (format nil "~A: Interrupted system call
should be
restarted" (string-downcase (format nil "~:A" command))))))
  (= x -86) (return (format nil "~A: Streams pipe error"
(string-downcase
(format nil "~:A" command))))))

  (= x -87) (return (format nil "~A: Too many users" (string-
downcase (format
nil "~:A" command))))))
  (= x -88) (return (format nil "~A: Socket operation on non-
socket" (string-
downcase (format nil "~:A" command))))))
  (= x -89) (return (format nil "~A: Destination address
required" (string-
downcase (format nil "~:A" command))))))
  (= x -90) (return (format nil "~A: Message too long"
(string-downcase
(format nil "~:A" command))))))
  (= x -91) (return (format nil "~A: Protocol wrong type for
socket" (string-
downcase (format nil "~:A" command))))))
  (= x -92) (return (format nil "~A: Protocol not available"
(string-downcase
(format nil "~:A" command))))))
  (= x -93) (return (format nil "~A: Protocol not supported"
(string-downcase
(format nil "~:A" command))))))
  (= x -94) (return (format nil "~A: Socket type not
supported" (string-
downcase (format nil "~:A" command))))))
  (= x -95) (return (format nil "~A: Operation not supported"
(string-
downcase (format nil "~:A" command))))))
  (= x -96) (return (format nil "~A: Protocol family not
supported" (string-
downcase (format nil "~:A" command))))))
  (= x -97) (return (format nil "~A: Address family not
supported by
protocol" (string-downcase (format nil "~:A" command))))))
  (= x -98) (return (format nil "~A: Address already in use"
(string-downcase
(format nil "~:A" command))))))
  (= x -99) (return (format nil "~A: Cannot assign requested
address"
(string-downcase (format nil "~:A" command))))))
  (= x -100) (return (format nil "~A: Network is down"
(string-downcase
(format nil "~:A" command))))))
  (= x -101) (return (format nil "~A: Network is unreachable"
(string-downcase
(format nil "~:A" command))))))
  (= x -102) (return (format nil "~A: Network dropped
connection because of
reset" (string-downcase (format nil "~:A" command))))))
  (= x -103) (return (format nil "~A: Software caused
connection abort"
(string-downcase (format nil "~:A" command))))))
  (= x -104) (return (format nil "~A: Connection reset by
peer" (string-
downcase (format nil "~:A" command))))))
  (= x -105) (return (format nil "~A: No buffer space
available" (string-

```

```

downcase (format nil "~:A" command))))
((= x -106) (return (format nil "~A: Transport endpoint is
already connected"
(string-downcase (format nil "~:A" command)))))
((= x -107) (return (format nil "~A: Transport endpoint is
not connected"
(string-downcase (format nil "~:A" command)))))
((= x -108) (return (format nil "~A: Cannot send after
transport endpoint
shutdown" (string-downcase (format nil "~:A" command)))))
((= x -109) (return (format nil "~A: Too many references:
cannot splice"
(string-downcase (format nil "~:A" command)))))
((= x -110) (return (format nil "~A: Connection timed out"
(string-downcase
(format nil "~:A" command)))))
((= x -111) (return (format nil "~A: Connection refused"
(string-downcase
(format nil "~:A" command)))))
((= x -112) (return (format nil "~A: Host is down" (string-
downcase (format
nil "~:A" command)))))
((= x -113) (return (format nil "~A: No route to host"
(string-downcase
(format nil "~:A" command)))))
((= x -114) (return (format nil "~A: Operation already in
progress" (string-
downcase (format nil "~:A" command)))))
((= x -115) (return (format nil "~A: Operation now in
progress" (string-
downcase (format nil "~:A" command)))))
((= x -116) (return (format nil "~A: Scale NFS file handle"
(string-downcase
(format nil "~:A" command)))))
((= x -117) (return (format nil "~A: Structure needs
cleaning" (string-
downcase (format nil "~:A" command)))))
((= x -118) (return (format nil "~A: Not a XENIX named type
file" (string-
downcase (format nil "~:A" command)))))
((= x -119) (return (format nil "~A: No XENIX semaphores
available" (string-
downcase (format nil "~:A" command)))))
((= x -120) (return (format nil "~A: Is a named type file"
(string-downcase
(format nil "~:A" command)))))
((= x -121) (return (format nil "~A: Remote I/O error"
(string-downcase
(format nil "~:A" command)))))
((= x -122) (return (format nil "~A: Disk quota exceeded"
(string-downcase
(format nil "~:A" command)))))
((= x -123) (return (format nil "~A: No medium found"
(string-downcase
(format nil "~:A" command)))))

((= x -124) (return (format nil "~A: Wrong medium type"
(string-downcase
(format nil "~:A" command)))))
;-----OBNOXIOUS RESPONSES-----
((= x -125) (return (format nil "~A: Memory Spillage has
occured! Please
clean up and try again." (string-downcase (format nil "~:A"
command)))))
((= x -126) (return (format nil "~A: General logic fault,
slipped on a 1! Did
you remember to clean up your memory spill?" (string-downcase (format
nil
 "~:A" command)))))
((= x -127) (return (format nil "~A: General logic fault,
slipped on a 0! Did
you remember to clean up your memory spill?" (string-downcase (format
nil
 "~:A" command)))))
((= x -128) (return (format nil "~A: .lib section in a.out
corrupted"
(string-downcase (format nil "~:A" command)))))
((= x -129) (return (format nil "~A: Argh! File not found!"
(string-downcase
(format nil "~:A" command)))))
((= x -130) (return (format nil "~A: We never had this
conversation, now
shred that document." (string-downcase (format nil "~:A" command)))))
((= x -131) (return (format nil "~A: Oh yeah, what's in it
for me?" (string-
downcase (format nil "~:A" command)))))
((= x -132) (return (format nil "~A: I don't wanna." (string-
downcase (format
nil "~:A" command)))))
((= x -133) (return (format nil "~A: Illegal byte sequence.
What are you
thinking?" (string-downcase (format nil "~:A" command)))))
((= x -134) (return (format nil "~A: File descriptor in bad
state. Looks like
you really screwed up this time!" (string-downcase (format nil "~:A"
command)))))
((= x -135) (return (format nil "~A: Would you like to play a
game?" (string-
downcase (format nil "~:A" command)))))
((= x -136) (return (format nil "~A:
50483333524D796C333354536B96313153. I
feel sick. =( " (string-downcase (format nil "~:A" command)))))
((= x -137) (return (format nil "~A: Logic dirty! Please wash
computer and,
try again." (string-downcase (format nil "~:A" command)))))
((= x -138) (return (format nil "~A: Fractal radiation
jamming the backbone."
(string-downcase (format nil "~:A" command)))))
((= x -139) (return (format nil "~A: I/O dam collapsed. Run
for your life!!!"
(string-downcase (format nil "~:A" command)))))

```

```

        ((= x -140) (return (format nil "~A: PC load letter."
(string-downcase
(format nil "~:A" command))))
        ((= x -141) (return (format nil "~A: General protection
fault! ...
Recovered." (string-downcase (format nil "~:A" command)))))
        ((= x -142) (return (format nil "~A: Kernel panic! ...
Recovered." (string-
downcase (format nil "~:A" command)))))
        -----DEFAULT RESPONSE-----
        (t (return (format nil "~A: Operation not permitted"
(string-downcase (format
nil "~:A" command)))))
    )
    (qinit)
    ;=====
    (defun first-string (list)
    (do
        ((x list (cdr x)) (res nil (format nil "~:A~:A" res (car x))))
        ((equal (car x) '#Space) res)
        (if (atom x) (return res))
        (if (null x) (return nil)))
    )
    )
    ;=====
    (defun car-string (string)
    (prog ()
        (setq list (coerce string 'list))
        (setq element (first-string list))
        (if (equal element nil) (return nil))
        (string-left-trim "()" element)
        (setq rest (string-left-trim " " (string-left-trim element
string)))
        (return rest)
    )
    )
    ;=====
    (defun cdr-string (string)
    (prog ()
        (setq list (coerce string 'list))
        (setq element (first-string list))
        (if (equal element nil) (return nil))
        (string-left-trim "()" element)
        (setq rest (string-left-trim " " (string-left-trim element
string)))
        (return rest)
    )
    )
    ;=====
    (defun brain () (progn (qinit) (loop (handler)) ))
    (defun subbrain () (progn (qinit) (loop (gget) (gsnd 0 "ok"))))
    // # File brain-null
    if test -f brain.fas
    then if test brain.lisp -ot brain.fas; then B="brain.fas"; else
B="brain.lisp"; fi
    else B="brain.lisp"; fi
    export B

echo "$B"
if test -f $1
then
    /usr/local/lib/clisp/base/qclisp/lisp.run -B /usr/local/lib/clisp \
-q -M /usr/local/lib/clisp/base/qclisp/lispinit.mem \
-i call-qfunc -i $B -x "(in-package 'QLISP)" (subbrain
"$1")"
    else
    echo "# No such file as $1"
fi
// # File brain-run
if test -f brain.fas
then if test brain.lisp -ot brain.fas; then B="brain.fas"; else
B="brain.lisp"; fi
else B="brain.lisp"; fi
export B
echo "$B"
if test -f $1
then
    /usr/local/lib/clisp/base/qclisp/lisp.run -B /usr/local/lib/clisp \
-q -M /usr/local/lib/clisp/base/qclisp/lispinit.mem \
-i call-qfunc -i $B -x "(in-package 'QLISP)" (brain "$1")"
    else
    echo "# No such file as $1"
fi
#include "clisp.h"
extern object module__call_qfunc__object_tab[];
subr_module__call_qfunc__subr_tab[1];
uintc module__call_qfunc__subr_tab_size = 0;
subr_initdata module__call_qfunc__subr_tab_initdata[1];
object module__call_qfunc__object_tab[1];
object_initdata module__call_qfunc__object_tab_initdata[1];
uintc module__call_qfunc__object_tab_size = 0;
extern int (qinit)();
extern char* (gget)();
extern int (gsnd)();
void module__qdestroy()();
void module__call_qfunc__init_function_1(module)
var module_* module;
{
    void module__call_qfunc__init_function_2(module)
var module_* module;
{
    register_foreign_function(&qinit, "qinit", 1024);
    register_foreign_function(&qget, "qget", 1024);
    register_foreign_function(&gsnd, "gsnd", 1024);
    register_foreign_function(&qdestroy, "qdestroy", 1024);
}
// # File call-qfunc.lisp
; Copyright - Sandia National Laboratories 2002
; By Chris Muejder
; lisp interface to qfunc.c
(DEFPACKAGE "QLISP" (:use "LISP" "FFI"))
(IN-PACKAGE "QLISP")
(export 'qinit)

```

```
(export 'qget)
(export 'qsnd)
(export 'qdestroy)
(def-c-call-out qinit (:return-type int))
(def-c-call-out qget (:return-type c-string))
(def-c-call-out qsnd (:arguments (code int) (msg c-string)) (:return-
type int))
(def-c-call-out qdestroy (:return-type int))
// # File maker
base=/usr/local/lib/clisp/base;export base
rm -rf $(base)/qlisp
export CLISP_LINKKIT=/usr/local/lib/clisp/linkkit/
rm -rf call-qfunc.c call-qfunc.o call-qfunc.lisp.fas qlisp.lib qlisp
qlispout
/usr/local/lib/clisp/clisp-link create-module-set qlisp call-qfunc.c
cc -O -c qfunc.c
cd qlisp
ln -s ../qfunc.o qfunc.o
cp -f ../thingtofix link.sh
cd ..
$(base)/lisp.run -M $(base)/lispinit.mem -q -c call-qfunc.lisp
/usr/local/lib/clisp/clisp-link add-module-set qlisp $(base)
$(base)/qlisp
echo "(exit)" | $(base)/qlisp/lisp.run -q -M
$(base)/qlisp/lispinit.mem -i call-qfunc
$(base)/qlisp/lisp.run -q -M $(base)/qlisp/lispinit.mem -q -i call-
qfunc -c brain.lisp
echo "(compile-file \"brain.lisp\")" | $(base)/qlisp/lisp.run -q -M
$(base)/qlisp/lispinit.mem -q -i call-qfunc
echo "Run with $(base)/qlisp/lisp.run -M $(base)/qlisp/lispinit.mem -i
call-qfunc.fas"
// # File qfunc.c
/* Copyright - Sandia National Laboratories 2002
* Functions to generate and destroy SysV queues, and to get/send data
to it
*/
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<syscall.h>
#include<errno.h>
#include<unistd.h>
/* #define __NR_ad 0
__syscall1(long,ad,int,foo); */
#define _MSG_SIZE_ 255
struct msgbuf
{int mtype;
int32_t opcode;
char mtext[_MSG_SIZE_];
};
int32_t qid=-1;
struct msgbuf qmsg;
char buf[_MSG_SIZE_];
int qinit() (qid=msgget((key_t) (12345), 0777 |
IPC_CREAT);return(qid);)

char * qget()
{FILE * f;
printf("getting\n");
while(1)
{
msgrcv(qid, &qmsg, sizeof(struct msgbuf), 0, MSG_NOERROR);
msgrcv(qid, &qmsg, sizeof(struct msgbuf), 1, MSG_NOERROR);
printf("got a message\n");
if(qmsg.mtype>65535) continue;
printf("pid=%d, code=%d, string=%s\n", qmsg.mtype, qmsg.opcode,
qmsg.mtext);
if(qmsg.mtype==1)
{
printf("its for me!\n");
//ad(qmsg.opcode);
//if(errno!=EAGAIN) continue;
memcpy(msg, qmsg.mtext, _MSG_SIZE_);
//set mtype
qmsg.mtype=qmsg.opcode;
}
else
{
//ad(qmsg.mtype);
//if(errno!=EAGAIN) msgsnd(qid, &qmsg, sizeof(struct
msgbuf), IPC_NOWAIT);
msgsnd(qid, &qmsg, sizeof(struct msgbuf), IPC_NOWAIT);
continue;
}
return qmsg.mtext;
}
}
int qsnd(int32_t code, char * msg)
{printf("Sending\n");
qmsg.opcode=code;memcpy(qmsg.mtext, msg, _MSG_SIZE_);
return(msgsnd(qid, &qmsg, sizeof(struct msgbuf), IPC_NOWAIT));
}
int qdestroy() (return(msgctl(qid, IPC_RMID, 0));)
/*int main(int argc, char**argv)
{
int code=0;
char msg[_MSG_SIZE_]="";
qinit();
while(1)
{
qget(&code, msg);
qsnd(42, "10.0.0.123");
}
qdestroy();
return 0;
}
/*EOF*/
// # File run
if test -f brain.fas
then if test brain.lisp -ot brain.fas; then B="brain.fas"; else
B="brain.lisp"; fi
else B="brain.lisp"; fi
```

```

export B
echo "$B"
if test "$0" = Z
then
    /usr/local/lib/csharp/base/qsharp/lisp.run -q \
    -M /usr/local/lib/csharp/base/qsharp/lispinit.mem \
    -i call-qfunc -i $B -x '(in-package "QLISP")'
else
    /usr/local/lib/csharp/base/qsharp/lisp.run -q \
    -M /usr/local/lib/csharp/base/qsharp/lispinit.mem \
    -i call-qfunc -i $B $*
fi

// # File thingofix
file_list='qfunc.o'
mod_list=''
if test -r call-qfunc.c; then
    file_list="$file_list" call-qfunc.o'
    mod_list="$mod_list" call_qfunc'
fi

make csharp-module CC="$CC" CFLAGS="$CFLAGS"
INCLUDES="$absolute_linkitdir"
NEW_FILES="$file_list"
NEW_LIBS="$file_list"
NEW_MODULES="$mod_list"
TO_LOAD=''

/* PLAC.go file 1
/* These 'PLAC.go' files perform automated startup of computers
running the "White Glove" Linux
distribution using deception kernel modifications according to
specific embodiments of the present
invention. These CD-based systems use these startup files so that the
deception mechanism begins to
operate at system startup. The is an example "PLAC.go" startup file
for a network deception that
redirects unauthorized programs to a secondary machine for execution.
*/
cp -af /mnt/floppy/ssh /etc
mkdir /root/.ssh/
cp -f /mnt/floppy/known_hosts /root/.ssh
service sshd start
webserver /cdrom-real/www
ifconfig eth0 10.0.30.1
#rm -rf /root/Trojan
#cp -f /mnt/floppy/wrapsSH.demo /root/wrap
# new for brains
cp -af /mnt/floppy/q2lisp /tmp/q2lisp
cd /tmp/q2lisp
chmod 700 maker; ./maker
cd base+q2lisp
cp /mnt/floppy/brain.1 .
cp /mnt/floppy/allowed.lisp .
./lisp.run -M lispinit.mem -i q2lisp.fas -i brain.1 > /dev/null
2>/dev/null&
cp -f /mnt/floppy/brainwrap /root/wrap
This is the "PLAC.go" startup file for the recipient system:
cp -af /mnt/floppy/ssh /etc

mkdir /root/.ssh/
cp -f known_hosts /root/.ssh
service sshd start
webserver /cdrom-real/www
ifconfig eth0 10.0.30.2
rm -rf /root/Trojan/
/* Example Wrapper files
The following includes all of the programs and build information
associated with an example
implementation of wrapper technology according to specific embodiments
of the present invention. Each
file is named on a line starting with '#'. The README file describes
the other files. In this example code,
different functions that can be performed by a wrapper are described
and enabled as different wrapXX.c
files. In this case, for example, wrapnull.c simply calls the
original exec with the original arguments,
after first making sure that it is not calling itself. It will be
understood for the teachings herein that a
single wrapper module will typically have several of the functions
illustrated by the sample wrapXX.c
code files below and in various implementations, all or most functions
may be encoded in a single code
file. */
// # File README
File name Description
-----
wrapper.h header file for all wrapper programs
wrapnull.c null wrapper - default install
-----
log.c logging and depth calculation support
log2.c old logging - kept in case
wraplog.c logging wrapper
-----
ACextract cat /dev/wraplog | ACextract > AC.h to build Access
Control
AC.c Access Control functions
AC.h Access Control table (autogen with ACextract)
wrapAC.c Access Control wrapper
-----
PLACextract cat /dev/wraplog | ACextract > PLAC.h to build PL
Access Control
PL.c Process lineage generation and logging functions
PLAC.c Process Lineage Access Control function
PLAC.h Process Lineage Access Control table (autogen with
PLACextract)
wrapPL.c Process Lineage wrapper
wrapPLlog.c Process Lineage logger (for use in making PLAC.h)
-----
wrapsSH.c SSH wrapper for 2-system deception
wrapNPS.c NPS wrapper program (AC + depth limit)
-----
wrapIPC.c IPC wrapper program
-----
Dextract Extract depths from wraplog (cat /root/wraplog | Dextract)
for D.h

```

```

argcount      Counts arguments (echo $#) in support of Dextract
D.C           Depth control functions
D.h           Depth assignments for pathname, depth pairs
wrapPLACD.C   Wrapper combining Process Lineage, Access Control,
              and Depth as controls
wrapD.C       Wrapper with exact depth as control
-----
NPSwrap.h     OLD - not necessary for a present embodiment
stuff.c       OLD - not necessary for a present embodiment
// # File AC.c
/* Access Control */
#include "AC.h"
/* Runs through list of allowed programs: 1 if 'filename' on list,
else 0 */
int ACalloved(char *filename)
{
    int i;
    for (i = 0; ALLOWED[i]; i++) {if (! strcmp(filename, ALLOWED[i]),
        strlen(filename))}
        return(1);}
    if (getpid() < SYSPROCTRESH) return(1); /* system processes ... we
    forgive them for
    now */
    return 0;
}
// # File ACextract
echo 'char *ALLOWED[] = {'
grep '/.' | sed 's:/:g' | while read a b; do echo "\"$a\""; done |
sort |
noreps.pl
echo "*/sbin/init"
echo '};'
// # File AC.h
char *ALLOWED[] = {
    "/bin/df",
    "/bin/cp",
    "/bin/ps",
    "/bin/sh",
    "/usr/bin/date",
    "/usr/bin/dv",
    "/usr/bin/find",
    "/usr/bin/sort",
    "/usr/bin/tail",
    "/usr/local/bin/me"
};
// # File argcount
echo -n $#
// # File BACextract
echo "(setq commands (make-hash-table :test 'equal))"
grep '/.' | sed 's:/:g' | while read a b; do echo "(self (gethash
\"$a\" commands)
'allowed\"; done | sort | noreps.pl
// # File D.C
/* Process Lineage Access Control */
#include "D.h"
/* Runs through list of allowed programs: 1 if 'filename' on list,
else 0 */
int DAllowed(char *filename)
{
    int i;
    for (i = 0; DALLOWED[i].name; i++)
        {if ((DALLOWED[i].depth == depth) &&
            (0 == strcmp(filename, DALLOWED[i].name, strlen(filename))))
            return(1);}
    if (getpid() < SYSPROCTRESH) return(1); /* system processes... we
    forgive them for
    now */
    return 0;
}
// # File Dextract
grep '/.' | sort | noreps.pl | while read a; do echo "\"$a\""; done | sed
's:/:g' |
while read a b
do
    echo -n "$a "
    echo 'argcount $b'
done
// # File D.h
struct dallow {char *name; int depth;};
void initdallow()
{
    DALLOWED[0].name="/bin/df";DALLOWED[0].depth=12;
    DALLOWED[1].name="/bin/cp";DALLOWED[1].depth=12;
    DALLOWED[2].name="/bin/ps";DALLOWED[2].depth=12;
    DALLOWED[3].name="/bin/sh";DALLOWED[3].depth=12;
    DALLOWED[4].name="/usr/bin/date";DALLOWED[4].depth=12;
    DALLOWED[5].name="/usr/bin/dv";DALLOWED[5].depth=12;
    DALLOWED[6].name="/usr/bin/find";DALLOWED[6].depth=12;
    DALLOWED[7].name="/usr/bin/sort";DALLOWED[7].depth=12;
    DALLOWED[8].name="/usr/bin/tail";DALLOWED[8].depth=12;
    DALLOWED[9].name="/usr/local/bin/me";DALLOWED[9].depth=12;
}
// # File Log2.c
/* Logs the programs that have been run.
* logfile: name of the file to put the log in
* program: name of the program being wrapped
* argc: number of args for program
* argv: argument array for program*/
int log(char *logfile, char *program, int argc, char *argv[])
{
    FILE *logfp;
    int i, pid;
    char pidstr[MAXPIDDIGITS];
    logfp = fopen(logfile, "a");
    if (! logfp)
        {
            //printf("Can't open log file\n");
            return -1;
        }
    fputs("\nUser's call\\", logfp);
    fputs(program, logfp);
    fputs("\n\\Args\\", logfp);
    for (i = 0; i < argc; i++)

```



```

fputs(", ", logfp);
fputs(argv[i], logfp);
}
pid = getpid();
itoa(pid, pidstr);
log_lineage(logfp, pidstr);
fputs("\n", logfp);
fclose(logfp);
return 1;
}

/*****
 * Traces the process lineage of process pid and logs the results in
 * file filep. Results are in the form of a .csv file.
 */
void log_lineage(FILE *filep, char *pid)
{
    FILE *pstats;
    char *statline, *pathname, *ppid, *stat, *label;
    pathname = (char *) malloc(strlen(PROC)+strlen(pid)+strlen(STATUS))
        * sizeof(char);
    if (! pathname)
    {
        printf("Insufficient memory for logging\n");
        return;
    }
    sprintf(pathname, "%s%s", PROC, pid, STATUS);
    pstats = fopen(pathname, "r");
    if (! pstats)
    {
        printf("Can't open file %s for reading\n", pathname);
        return;
    }
    free(pathname);
    statline = (char *) malloc(MAXLINELEN * sizeof(char));
    label = (char *) malloc(MAXLABELLEN * sizeof(char));
    stat = (char *) malloc(MAXINFOLEN * sizeof(char));
    ppid = (char *) malloc(MAXPIDDIGITS * sizeof(char));
    if (! (statline) || ! (label) || ! (stat) || ! (ppid))
    {
        printf("Insufficient memory for logging\n");
        return;
    }
    while (! fgetc(statline, MAXLINELEN, pstats))
    {
        if (! strcmp(ppid, statline, strlen(ppid)))
        {
            getlabel(statline, label);
            getinfo(statline, ppid);
            fputs("\n", filep);
            fputs(label, filep);
            fputs("\n", filep);
            fputs(ppid, filep);
        }
        else
        {
            getlabel(statline, label);
            getinfo(statline, stat);
            fputs("\n", filep);
            fputs(label, filep);
            fputs("\n", filep);
            fputs(stat, filep);
            fputs("\n", filep);
        }
    }
}

/*****
 * Extracts the data from a line of the status
 * file in proc. Lines are of the form:
 * "Label: data"
 * procline: a line of the file /proc/(pid)/status
 * infobuf: array to put the data in*/
void getinfo(char *procline, char *infobuf)
{
    int i, j, getdata, logspaces;
    getdata = logspaces = j = 0;
    for (i = 0; i < strlen(procline); i++)
    {
        if (getdata)
        {
            if (logspaces && (procline[i] != '\n'))
            {
                infobuf[j] = procline[i];
                j++;
            }
            else if (! isspace(procline[i]))
            {
                logspaces = 1;
                infobuf[j] = procline[i];
                j++;
            }
            else if (procline[i] == ':')
            {
                getdata = 1;
            }
            infobuf[j] = (char) NULL;
        }
    }
}

/*****
 * Extracts the label and from a line of the status
 * file in proc. Lines are of the form:

```

```

* "Label: data"
*
* procline: a line of the file /proc/(pid)/status
* infobuf: array to put the data in*/
int getlabel(char *procline, char *infobuf)
{
    int i, getdata;
    getdata = i = 0;
    while (procline[i] != ':')
    {
        infobuf[i] = procline[i];
        i++;
    }
    infobuf[i] = (char) NULL;
    return i;
}

/* From K&R page 64 (see wrapper.h for details)
* Converts an int n to a string, puts the results in s.*/
void itoa(int n, char s[])
{
    int i, sign;
    if ((sign = n) < 0)
    {
        n = -n;
    }
    i = 0;
    do
    {
        s[i++] = n % 10 + '0';
    }
    while ((n /= 10) > 0);
    if (sign < 0)
    {
        s[i++] = '-';
    }
    s[i] = '\0';
    reverse(s);
}

/* From K&R page 62 (see wrapper.h for details)
* Reverses the order of the characters in s.*/
void reverse(char s[])
{
    int c, i, j;
    for (i = 0, j = strlen(s) - 1; i < j; i++, j--)
    {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

// # File log.c
#define DEPTH_THRESH 16
int depth;

/* Log the programs that have been run. */
int log(char *logfile, char *program, int argc, char *argv[])
{
    FILE *logfp;

    int i, err, pid;
    char pidstr[MAXPIDDIGITS];
    logfp = fopen(logfile, "a");
    if (!logfp) {return -1;}
    fputs("\nUser's call\n", logfp);
    fputs(program, logfp);
    for (i = 0; i < argc; i++) {fputs(" ", logfp); fputs(argv[i], logfp);}
    pid = getpid();
    itoa(pid, pidstr);
    if ((err=loglineage(logfp, pidstr)) != 0) fputs("\n", logfp);
    fclose(logfp);
    return 1;
}

/*****
* Traces the process lineage of process pid and logs the results in
* file filep. Results are in the form of a .csv file.
**/
int lineage(FILE *filep, char *pid)
{
    FILE *pstats;
    char *statline, *pathname, *pid, *stat, *label;
    pathname = (char *) malloc((strlen(PROC)+strlen(pid)+strlen(STATUS)) *
        sizeof(char));
    if (!pathname) {printf("Insufficient memory for logging\n");return
        1;}
    sprintf(pathname, "%s%s", PROC, pid, STATUS);
    pstats = fopen(pathname, "r");
    if (!pstats) {printf("Can't open file %s for reading\n",
        pathname);return 1;}
    free(pathname);
    statline = (char *) malloc(MAXLINELEN * sizeof(char));
    label = (char *) malloc(MAXLABELLEN * sizeof(char));
    stat = (char *) malloc(MAXINFOLEN * sizeof(char));
    pid = (char *) malloc(MAXPIDDIGITS * sizeof(char));
    if (!statline) || (!label) || (!stat) || (!pid)
        {printf("Insufficient memory for logging\n");return 1;}
    fputs("\n->\n", filep);
    while (fgets(statline, MAXLINELEN, pstats)
        {if (! strcmp(PPID, statline, strlen(PPID)))
            {
                getlabel(statline, label);
                getinfo(statline, pid);
                fputs("\n", filep);
                fputs(label, filep);
                fputs("\n", filep);
                else {getlabel(statline, label);
                    getinfo(statline, stat);
                    fputs("\n", filep);
                    fputs(label, filep);
                    fputs("\n", filep);
                    fputs(stat, filep);
                    fputs("\n", filep);
                }
            }
        }
    if (strcmp(pid, ZERO, strlen(ZERO)) {depth++;loglineage(filep,
        pid);}
}

```

```

fclose(pstats); free(statline); free(stat); free(label); free(ppid);
return (0);
}
/* Extract data from proc */
void getinfo(char *procline, char *infobuf)
{
    int i, j, getdata, logspaces;
    getdata = logspaces = j = 0;
    for (i = 0; i < strlen(procline); i++)
        (if (getdata)
            (if (logspaces && (procline[i] != '\n'))
                (infobuf[j] = procline[i]; j++;)
            else if (! isspace(procline[i]))
                (logspaces = 1;
                 infobuf[j] = procline[i];
                 j++;)
            )
        )
        else if (procline[i] == ':') (getdata = 1;)
    }
    infobuf[j] = (char) NULL;
}
/* Extracts the label and from a line of the status
 * file in proc. Lines are of the form:
 * * "Label: data"
 *
 * procline: a line of the file /proc/(pid)/status
 * infobuf: array to put the data in*/
int getlabel(char *procline, char *infobuf)
{
    int i, getdata;
    getdata = i = 0;
    while (procline[i] != ':') (infobuf[i] = procline[i]; i++;)
    infobuf[i] = (char) NULL;
    return i;
}
/* From K&R page 64 (see wrapper.h for details)
 * Converts an int n to a string, puts the results in s. */
void itoa(int n, char s[])
{
    int i, sign;
    if ((sign = n) < 0) (n = -n;)
    i = 0;
    do {s[i++] = n % 10 + '0';} while ((n /= 10) > 0);
    if (sign < 0) (s[i++] = '-');
    s[i] = '\0';
    reverse(s);
}
/* From K&R page 62 (see wrapper.h for details)
 * Reverses the order of the characters in s. */
void reverse(char s[])
{
    int c, i, j;
    for (i = 0, j = strlen(s) - 1; i < j; i++, j--)
        (c = s[i]; s[i] = s[j]; s[j] = c;)
}
/* # File Makefile
all: wrapnull wraplog wrapIPC wrapNPS wrapSSH wrapPLlog wrapPL
wrapPLACD wrapD wrapBrain

```

```

wrapnull: wrapper.h wrapnull.c
gcc -Wall -o wrapnull wrapnull.c
wraplog: wrapper.h wraplog.c log.c
gcc -Wall -o wraplog wraplog.c
wrapAC: wrapper.h wrapAC.c PL.c AC.c AC.h
gcc -Wall -o wrapAC wrapAC.c
wrapD: wrapper.h wrapD.c PL.c D.c D.h
gcc -Wall -o wrapD wrapD.c
wrapIPC: wrapper.h ../brains/IPctest.c wrapIPC.c
gcc -Wall -o IPctest ../brains/IPctest.c
wrapBrain: wrapper.h ../brains/IPctest.c wrapBrain.c
gcc -Wall -o wrapBrain wrapBrain.c
cd ../brains; make brain
wrapNPS: NPSwrap.h wrapNPS.c log.c AC.c AC.h
gcc -Wall -o wrapNPS wrapNPS.c
wrapSSH: wrapper.h wrapSSH.c AC.c AC.h
gcc -Wall -o wrapSSH wrapSSH.c
wrapPLlog: wrapper.h wrapPLlog.c PL.c pathname.c
gcc -Wall -o wrapPLlog wrapPLlog.c
wrapPL: wrapper.h wrapPL.c PL.c PLAC.c PLAC.h pathname.c
gcc -Wall -o wrapPL wrapPL.c
wrapPLACD: wrapper.h wrapPLACD.c AC.c PL.c PLAC.c PLAC.h pathname.c
gcc -Wall -o wrapPLACD wrapPLACD.c
clean:
touch *. [ch]
fresh:
rm -f nullwrap logwrap IPCwrap NPSwrap SSHwrap
// # File NPSwrap.h
#define WRAPPER "/root/wrap"
#define LOG_FILE "/root/wraplog.csv"
// For tracking process lineage
#define PROC "/proc/"
#define STATUS "/status"
#define MAXPIDDIGITS 10
#define MAXLINELEN 500
#define MAXINFOLLEN 100
#define MAXLABELLEN 50
#define PPID "PPid"
#define ZERO "0"
// For the su demo
#define DEMO 0 /* If DEMO = 1, run the su demo */
#define SU "/bin/su"
#define SU_FILE "/tmp/sucount"
#define FAKE_SU "/root/su"
/* Access control variables */
#define DEPTH_THRESH 16
/* Executable names */
/* Access control functions */
int is_allowed(char *filename);
/* Taken from "The C Programming Language" by Kernighan & Ritchie
 * Pages 62 and 64 of the second edition*/
// # File pathname.c
#define MAXPATHLEN 1024
char pathname[MAXPATHLEN];
int getpathname(char *program)

```

```
(int pid;
int pipefd[2];
int waitstat;
if (0 != pipe(pipefd)) {return PIPERR;}
if (0 != (pid = fork())) /* Parent or failed */
{if (pid < 0) {close(pipefd[0]); close(pipefd[1]); return FORKERR;}
if (waitpid(pid, &waitstat, 0) != pid) {return WAITERR;}
if (WEXITSTATUS(waitstat) == 0)
{read(pipefd[0], &pathname, MAXPATHLEN); close(pipefd[0]);
close(pipefd[1]);}
else {close(pipefd[0]); close(pipefd[1]); return GETWDERR;}
}
else { /* Child */
int i;
char dirname[MAXPATHLEN];
close(pipefd[0]);
for (i = strlen(program); i >= 0; i--)
{if (program[i] == '/')
{program[i] = '\0';
if (0 != chdir(program)) {exit(1);}
if (0 == getcwd(dirname, MAXPATHLEN)) {exit(1);}
break;}
}
if (i == -1) {if (0 == getcwd(dirname, MAXPATHLEN)) {exit(1);} }
strncat(dirname, "/", 1);
strncat(dirname, &(program[i+1]), MAXPATHLEN);
write(pipefd[1], &dirname, strlen(dirname)+1);
close(pipefd[1]);
exit(0);
}
return(0);
}
// # File PLAC.c
/* Process Lineage Access Control */
#include "PLAC.h"
/* Runs through list of allowed programs: 1 if 'filename' on list,
else 0 */
int Plallowed(char *filename)
{int i;
for (i = 0; PLALLOWED[i]; i++) {if (! strcmp(filename, PLALLOWED[i],
strlen(filename))) return(1);}
if (getuid() < SYSPROCTHRESH) return(1); /* system processes... we
forgive chem for
now */
return 0;
}
// # File PLACextrac
echo 'char *PLALLOWED[] = {'
grep '/' | sort | nreps.pl | while read a; do echo "\"$a\""; done
echo '}';
// # File PLAC.h
char *PLALLOWED[] = {
/bin/bash;wrap;xterm;blackbox;xinit;startx;bash;init",
/bin/cat;wrap;Wgnpulse.sh;blackbox;xinit;startx;bash;init",
"/bin/cat;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/cat;wrap;nessus-adduser.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/cat;wrap;nessus-mkcert.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/cat;wrap;ptadmn.pl;blackbox;xinit;startx;bash;init",
"/bin/cat;wrap;sh;ptadmn.pl;blackbox;xinit;startx;bash;init",
"/bin/cmod;wrap;Wgnpulse.sh;blackbox;xinit;startx;bash;init",
"/bin/cmod;wrap;nessus-
adduser.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/cmod;wrap;nessus-mkcert.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/chown;wrap;Wgnpulse.sh;blackbox;xinit;startx;bash;init",
"/bin/cp;wrap;Wgnpulse.sh;blackbox;xinit;startx;bash;init",
"/bin/cp;wrap;nessus-adduser.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/cp;wrap;sh;blackbox;xinit;startx;bash;init",
"/bin/df;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/df;wrap;ptadmn.pl;blackbox;xinit;startx;bash;init",
"/bin/df;wrap;sh;blackbox;xinit;startx;bash;init",
"/bin/df;wrap;sysvluw.pl;blackbox;xinit;startx;bash;init",
"/bin/dircolors;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/egrep;wrap;nessus-mkcert.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/grep;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/grep;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/grep;wrap;ifup-post;sh;blackbox;xinit;startx;bash;init",
"/bin/grep;wrap;ifup;sh;blackbox;xinit;startx;bash;init",
"/bin/grep;wrap;ifup;sh;blackbox;xinit;startx;bash;init",
"/bin/grep;wrap;sh;blackbox;xinit;startx;bash;init",
"/bin/hostame;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/hostame;wrap;ifup-post;sh;blackbox;xinit;startx;bash;init",
"/bin/hostame;wrap;ifup;sh;blackbox;xinit;startx;bash;init",
"/bin/hostname;wrap;nessus-
mkcert.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/hostname;wrap;ptadmn.pl;blackbox;xinit;startx;bash;init",
"/bin/id;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/id;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/id;wrap;nessus-adduser;nessus-
adduser.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/id;wrap;nessus-mkcert;nessus-
mkcert.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/less;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/lis;wrap;bash;xterm;blackbox;xinit;startx;bash;init",
"/bin/mkdir;wrap;Wgnpulse.sh;blackbox;xinit;startx;bash;init",
"/bin/mkdir;wrap;nessus-
adduser.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/mkdir;wrap;nessus-mkcert.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/mkdir;wrap;sh;sh;blackbox;xinit;startx;bash;init",
"/bin/mv;wrap;Wgnpulse.sh;blackbox;xinit;startx;bash;init",
"/bin/mv;wrap;nessus-mkcert.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/ps;wrap;ptadmn.pl;blackbox;xinit;startx;bash;init",
"/bin/rm;wrap;Wgnpulse.sh;blackbox;xinit;startx;bash;init",
"/bin/rm;wrap;ifup-post;sh;blackbox;xinit;startx;bash;init",
"/bin/rm;wrap;nessus-adduser.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/rm;wrap;nessus-mkcert.sh;sh;blackbox;xinit;startx;bash;init",
"/bin/sed;wrap;Wgnpulse.sh;blackbox;xinit;startx;bash;init",
"/bin/sed;wrap;ifup-alises;sh;blackbox;xinit;startx;bash;init",
"/bin/sed;wrap;ifup-alises;ifup-
post;sh;blackbox;xinit;startx;bash;init",
"/bin/sed;wrap;ifup-post;sh;blackbox;xinit;startx;bash;init",
post;sh;blackbox;xinit;startx;bash;init",
```

```

"/bin/sed;wrap;ifup;ifup;sh;blackbox;xinit;startx;bash;init",
"/bin/sed;wrap;nessus-adduser;nessus-
address;sh;sh;blackbox;xinit;startx;bash;init",
"/bin/sed;wrap;nessus-mkcert;sh;sh;blackbox;xinit;startx;bash;init",
"/bin/sh;wrap;blackbox;xinit;startx;bash;init",
"/bin/sh;wrap;ptadmin.pl;blackbox;xinit;startx;bash;init",
"/bin/sleep;wrap;sh;blackbox;xinit;startx;bash;init",
"/bin/touch;wrap;nessus-mkcert;sh;sh;blackbox;xinit;startx;bash;init",
"/bin/tput;wrap;bash;bash;xterm;blackbox;xinit;startx;bash;init",
"/cdrom/X11R6/bin/mozilla/mozilla-bin;wrap;run-mozilla.sh;init",
"/cdrom/X11R6/bin/mozilla/mozilla-bin;wrap;run-mozilla.sh;sh;blackbox;xinit;startx;bash;init",
"/cdrom/X11R6/bin/mozilla/mozilla;wrap;sh;blackbox;xinit;startx;bash;init",
"/etc/sysconfig/network-scripts/ifup-aliases;wrap;ifup-
post;sh;blackbox;xinit;startx;bash;init",
"/etc/sysconfig/network-scripts/ifup-
post;wrap;sh;blackbox;xinit;startx;bash;init",
"/etc/sysconfig/network-scripts/ifup-routes;wrap;ifup-
post;sh;blackbox;xinit;startx;bash;init",
"/sbin/ifconfig;wrap;ifup-aliases;ifup-aliases;ifup-
post;sh;blackbox;xinit;startx;bash;init",
"/sbin/ifconfig;wrap;ifup;sh;blackbox;xinit;startx;bash;init",
"/sbin/modprobe;wrap;devfsd;init",
"/sbin/modprobe;wrap;mozilla-bin;run-mozilla.sh;init",
"/sbin/modprobe;wrap;mozilla-bin;run-
mozilla.sh;sh;blackbox;xinit;startx;bash;init",
"/tmp/mpulse/autorecheck.pl;wrap;start;Wgmpulse.sh;blackbox;xinit;star
tx;bash;init",
"/tmp/mpulse/etc/start;wrap;Wgmpulse.sh;blackbox;xinit;startx;bash;ini
t",
"/tmp/mpulse/miniserv.pl;wrap;start;Wgmpulse.sh;blackbox;xinit;startx;
bash;init",
"/tmp/mpulse/mrelycd.pl;wrap;init",
"/usr/X11R6/bin/mozilla/run-mozilla.sh;wrap;init",
"/usr/X11R6/bin/mozilla/run-
mozilla.sh;wrap;sh;blackbox;xinit;startx;bash;init",
"/usr/X11R6/bin/xkill;wrap;blackbox;xinit;startx;bash;init",
"/usr/X11R6/bin/xsetroot;wrap;blackbox;xinit;startx;bash;init",
"/usr/X11R6/bin/xterm;wrap;sh;blackbox;xinit;startx;bash;init",
"/usr/X11R6/bin/xterm;wrap;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/basename;wrap;ifup-post;ifup-
post;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/basename;wrap;ifup-post;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/basename;wrap;ifup;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/basename;wrap;ifup;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/basename;wrap;ifup;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/basename;wrap;run-mozilla.sh;init",
"/usr/bin/basename;wrap;run-
mozilla.sh;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/clear;wrap;nessus-
mkcert;sh;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/diname;wrap;mozilla;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/diname;wrap;mozilla;sh;blackbox;xinit;startx;bash;init",
"/usr/bin/diname;wrap;run-mozilla.sh;init",

```

```

"/usr/bin/diname;wrap;run-
mozilla.sh;sh;blackbox:xinit;startx;bash;init",
"/usr/bin/find;wrap;Wgmpulise.sh;Wgmpulise.sh;blackbox:xinit;startx;bash
;init",
"/usr/bin/ipcalc;wrap;ifup;sh;blackbox:xinit;startx;bash;init",
"/usr/bin/kill;wrap;ptadmin.pl;blackbox:xinit;startx;bash;init",
"/usr/bin/netstat;wrap;netview.pl;blackbox:xinit;startx;bash;init",
"/usr/bin/port;wrap;Wgmpulise.sh;blackbox:xinit;startx;bash;init",
"/usr/bin/setty;wrap;bash;xterm;blackbox:xinit;startx;bash;init",
"/usr/bin/top;wrap;sysview.pl;blackbox:xinit;startx;bash;init",
"/usr/bin/uptime;wrap;sysview.pl;blackbox:xinit;startx;bash;init",
"/usr/bin/w;wrap;sh;ptadmin.pl;blackbox:xinit;startx;bash;init",
"/usr/local/bin/Wgmpulise.sh;wrap;blackbox:xinit;startx;bash;init",
"/usr/local/bin/awk;wrap;sh;ptadmin.pl;blackbox:xinit;startx;bash;init
",
"/usr/local/bin/ghtcrack;wrap;blackbox:xinit;startx;bash;init",
"/usr/local/bin/netview.pl;wrap;blackbox:xinit;startx;bash;init",
"/usr/local/bin/noreps.pl;wrap;bash;xterm;blackbox:xinit;startx;bash;
ash;init",
"/usr/local/bin/ptadmin.pl;wrap;blackbox:xinit;startx;bash;init",
"/usr/local/bin/services.pl;wrap;blackbox:xinit;startx;bash;init",
"/usr/local/bin/sysview.pl;wrap;blackbox:xinit;startx;bash;init",
"/usr/local/bin/time.pl;wrap;blackbox:xinit;startx;bash;init",
"/usr/local/bin/which;wrap;run-mozilla.sh;init",
"/usr/local/bin/which;wrap;run-
mozilla.sh;sh;blackbox:xinit;startx;bash;init",
"/usr/sectools/bin/nessus-w;wrap;sh;blackbox:xinit;startx;bash;init",
"/usr/sectools/bin/imapfe;wrap;blackbox:xinit;startx;bash;init",
"/usr/sectools/bin/openssl;wrap;nessus-
mkcert;sh;sh;blackbox:xinit;startx;bash;init",
"/usr/sectools/sbin/nessus-
adduser;wrap;sh;sh;blackbox:xinit;startx;bash;init",
"/usr/sectools/sbin/nessus-
mkcert;wrap;sh;sh;blackbox:xinit;startx;bash;init",
"/usr/sectools/sbin/nessusd;wrap;init",
"/usr/sectools/sbin/nessusd;wrap;nessus-
adduser;sh;sh;blackbox:xinit;startx;bash;init",
"/usr/sectools/sbin/nessusd;wrap;nessus-
adduser;sh;sh;blackbox:xinit;startx;bash;init",
"/usr/sectools/sbin/nessusd;wrap;nessus-
mkcert;sh;sh;blackbox:xinit;startx;bash;init",
);
// # File PL.c
#include <sys/wait.h>
#include "pathname.c"
#define MAXPLEN 8192
char lineage[MAXPLEN]; /* the last retrieved process lineage string
*/
#define LOGIT(x) logfp=fopen(LOG_FILE, "a");if (logfp)
(fputs(x,logfp);fputs(linage,logfp);fclose(logfp);} else
fprintf(stderr,"%s%s\n",x,linage);
#define DEPTH_THRESH 16
int depth;
/* Generate process lineage log entry in lineage

```

```

* Ignores possibility of '/' in a filename (not permitted by the
kernel)
* Ignores issues of SYM links in path */
int plog(char *program, int argc, char *argv[])
{char pidstr[MAXPIDDIGITS];
 int pid,err;
 err=0;depth=0;
 if (l == needpathname(program)) err=getpathname(program);
 else (strcpy(pathname, program, MAXPATHLEN));
 if (err != 0) return(err);
 translate(pathname); /* appends information into lineage for logging
 */
 pid = getpid();itoa(pid, pidstr);
 /* add PID to lineage */
 /*(char tmp[MAXPATHLEN]);
 // strcpy(tmp,pidstr,8);strncat(tmp, " ", 2);
 // strncat(tmp,pathname,MAXPATHLEN-
 10);strcpy(lineage,tmp,MAXPATHLEN);
 err=log_lineage(pidstr);
 if (err == 0) return(1); else return(err);
 }
 /* Looks for '.' and '..' in the pathname. (or not absolute)
 * Returns 1 if it finds some dots and 0 otherwise. */
 int i;
 if (path[0] != '/') {return(1);}
 if (path[0] == '/') {if ((path[1] == '.') && (path[2] == '/') ||
 (path[1] == '/'))
 {return 1;}}
 for (i = 0; i < strlen(path); i++)
 {if ((path[i] == '/') && (path[i+1] == '.'))
 {i += 2;
 if ((path[i] == '/') || ((path[i] == '.') && (path[i+1] ==
 '/')) {return 1;}}
 }
 }
 return 0;
 }
 /* Logs 'stat' into lineage, translating nonprintables into \xx
 * e.g., "x,y" is logged as "x\3by" */
 void translate(char *stat)
 {int i, gotbackslash;
 char tmpstring[2];
 gotbackslash = 0;
 for (i = 0; i < strlen(stat); i++)
 {if (((int) stat[i] > 126) || ((int) stat[i] < 33) ||
 (stat[i] == '\') || (stat[i] == ','))
 {char asciival[MAXASCII];
 if ((stat[i] == '\') && (! gotbackslash))
 {sprintf(asciival, "\\&x", (int) stat[i]);
 strncat(lineage, asciival, MAXPATHLEN);gotbackslash = 1;
 }
 else if ((stat[i] == '\') && (gotbackslash))
 { /* Nolog extra backslash */ gotbackslash = 0;
 }
 else {sprintf(asciival, "\\&x", (int) stat[i]);
 strncat(lineage, asciival, MAXPATHLEN);
 }
 }
 }
 }
 else {tmpstring[0]=stat[i];tmpstring[1]='\0';strncat(lineage,
 tmpstring,
 MAXPATHLEN);
 }
 }
 /* Trace process lineage, results in lineage ad a PATH-like string */
 int log_lineage(char *pid)
 {FILE *pstats;
 char *statline, *pathname, *pid, *stat, *label;
 pathname = (char *) malloc((strlen(PROC)+strlen(pid)+strlen(STATUS)) *
 sizeof(char));
 if (! pathname) {return(NOMEMERR);}
 if (! pathname) {return(NOMEMERR);}
 depth++;
 sprintf(pathname, "%s%s", PROC, pid, STATUS);
 pstats = fopen(pathname, "r");
 if (! pstats) {return(NOLOGERR);}
 free(pathname);
 statline = (char *) malloc(MAXLINELEN * sizeof(char));
 label = (char *) malloc(MAXLABELLEN * sizeof(char));
 stat = (char *) malloc(MAXFOLEN * sizeof(char));
 pid = (char *) malloc(MAXPIDDIGITS * sizeof(char));
 if (! statline) || (! label) || (! stat) || (! pid))
 {return(NOLOGERR);}
 while (!gets(statline, MAXLINELEN, pstats))
 {if (! strcmp(pid, statline, strlen(pid)))
 {getlabel(statline, label);getinfo(statline, pid);}
 else {getlabel(statline, label);getinfo(statline, stat);}
 if (! strcmp(NAME, statline, strlen(NAME)))
 {strncat(lineage, " ", 2);translate(stat);}
 }
 }
 if (strcmp(pid, ZERO, strlen(ZERO))) {log_lineage(pid);}
 else {strncat(lineage, "\n", 2);}
 fclose(pstats);
 free(statline);free(stat);free(label);free(pid);
 return(0);
 }
 /* Extract data from status lines in proc of the form: "label: data"
 */
 void getinfo(char *procline, char *infobuf)
 {int i, j, getdata, logspaces;
 getdata = logspaces = j = 0;
 for (i = 0; i < strlen(procline); i++)
 {if (getdata)
 {if (logspaces && (procline[i] != '\n'))
 {infobuf[j] = procline[i];j++;}
 else if (! isspace(procline[i]))
 {logspaces = 1;
 infobuf[j] = procline[i];
 j++;}
 }
 }
 else if (procline[i] == ':') {getdata = 1;
 }
 }
 infobuf[j] = (char) NULL;
 /* Extract label from status file in proc of the form: "label: data"
 */

```

```

int getlabel(char *procline, char *infobuf)
{int i, getdata;
  getdata = i = 0;
  while (procline[i] != ':') {infobuf[i] = procline[i], i++;}
  infobuf[i] = (char) NULL;
  return i;
}

/* From K&R page 64: int to string, results in s. */
void itoa(int n, char s[])
{int i, sign;
  if ((sign = n) < 0) {n = -n;}
  i = 0;
  do {s[i++] = n % 10 + '0';} while ((n /= 10) > 0);
  if (sign < 0) {s[i++] = '-';}
  s[i] = '\0';
  reverse(s);
}

/* From K&R page 62: Reverse order of characters in s. */
void reverse(char s[])
{int c, i, j;
  for (i = 0, j = strlen(s) - 1; i < j; i++, j--) {c = s[i]; s[i] =
s[j]; s[j] = c;}
}

// # File stuff.c
/*****
 * Logs the programs that have been run.
 *
 * logfile: name of the file to put the log in
 * program: name of the program being wrapped
 * argc: number of args for program
 * argv: argument array for program*/
int log(char *logfile, char *program, int argc, char *argv[]) {
  FILE *logfile;
  int i, pid;
  char *pidstr;
  logfile = fopen(logfile, "a");
  if (! logfile) {
    //printf("Can't open log file\n");
    //fclose(logfile);
    return -1;
  }
  fputs("\nUser's call\n", logfile);
  fputs(program, logfile);
  fputs("\n\nArgs\n", logfile);
  for (i = 0; i < argc; i++) {
    fputs(" ", logfile);
    fputs(argv[i], logfile);
  }
  pid = getpid();
  pidstr = (char *) malloc(sizeof(char) * MAXPIDDIGITS);
  itoa(pid, pidstr);
  log_lineage(logfile, pidstr);
  fputs("\n", logfile);
  fclose(logfile);
  return 1;
}

/*****
 * Traces the process lineage of process pid and logs the results in
 * file filep. Results are in the form of a .csv file.
 */
void log_lineage(FILE *filep, char *pid) {
  FILE *pstats;
  char *statline, *proc, *pathname, *ppid, *stat, *label;
  proc = (char *) malloc(strlen(PROC)+strlen(pid)*sizeof(char));
  strcpy(proc, PROC, strlen(PROC));
  pstat = strcat(proc, pid, strlen(pid));
  pathname = (char *) malloc(strlen(PROC) + strlen(STATUS)) *
sizeof(char);
  strcpy(pathname, proc, strlen(proc));
  pathname = strcat(pathname, STATUS, strlen(STATUS));
  pstats = fopen(pathname, "r");
  if (! pstats) {
    printf("Can't open file %s for reading\n", pathname);
    //fclose(pstats);
    return;
  }
  statline = (char *) malloc(MAXLINELEN * sizeof(char));
  label = (char *) malloc(MAXLABELLEN * sizeof(char));
  stat = (char *) malloc(MAXINFOLEN * sizeof(char));
  fputs("\n\n->\n", filep);
  while (fgets(statline, MAXLINELEN, pstats)) {
    if (! strcmp(ppid, statline, strlen(ppid))) {
      ppid = (char *) malloc(MAXPIDDIGITS * sizeof(char));
      getlabel(statline, label);
      getinfo(statline, ppid);
      fputs("\n", filep);
      fputs(label, filep);
      fputs("\n", filep);
      fputs(ppid, filep);
    }
    else {
      getlabel(statline, label);
      getinfo(statline, stat);
      fputs("\n", filep);
      fputs(label, filep);
      fputs("\n", filep);
      fputs(stat, filep);
      fputs("\n", filep);
    }
  }
  #ifdef DEPTH
  if (strcmp(ppid, ZERO, strlen(ZERO))) {depth++; log_lineage(filep,
ppid);}
  #endif
  if (strcmp(ppid, ZERO, strlen(ZERO))) {log_lineage(filep, ppid);}
  #endif
  fclose(pstats);
  free(statline);
  free(pathname);
  free(proc);
  free(stat);
  free(ppid);
}

```

```

)
/****
* Extracts the data from a line of the status
* file in proc. Lines are of the form:
* "Label: data"
*
* procline: a line of the file /proc/(pid)/status
* infobuf: array to put the data in*/
void getinfo(char *procline, char *infobuf) {
    int i, j, getdata, logspaces;
    getdata = logspaces = j = 0;
    for (i = 0; i < strlen(procline); i++) {
        if (getdata) {
            if (logspaces && (procline[i] != '\n')) {
                infobuf[j] = procline[i];
                j++;
            }
            else if (!isspace(procline[i])) {
                logspaces = 1;
                infobuf[j] = procline[i];
                j++;
            }
            else if (procline[i] == ':') {
                getdata = 1;
            }
        }
        infobuf[j] = (char) NULL;
    }
/****
* Extracts the label and from a line of the status
* file in proc. Lines are of the form:
* "Label: data"
*
* procline: a line of the file /proc/(pid)/status
* infobuf: array to put the data in*/
void getlabel(char *procline, char *infobuf) {
    int i, getdata, logspaces;
    getdata = logspaces = i = 0;
    while (procline[i] != ':') {
        infobuf[i] = procline[i];
        i++;
    }
    infobuf[i] = (char) NULL;
}

/* From K&R page 64 (see wrapper.h for details)
* Converts an int n to a string, puts the results in s.*/
void itoa(int n, char s[]) {
    int i, sign;
    if ((sign = n) < 0) {
        n = -n;
    }
    i = 0;
    do {
        s[i++] = n % 10 + '0';
    } while ((n /= 10) > 0);
}

    if (sign < 0) {
        s[i++] = '-';
    }
    s[i] = '\0';
    reverse(s);
}

/* From K&R page 62 (see wrapper.h for details)
* Reverses the order of the characters in s.*/
void reverse(char s[]) {
    int c, i, j;
    for (i = 0, j = strlen(s) - 1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
/****
* Keeps track of how many times "su" has been called in a row.*/
int process_su(char *command, char *argv[], char *envp[]) {
    FILE *suftp;
    int count;
    suftp = fopen(SU_FILE, "r");
    if (!suftp) {
        fclose(suftp);
        suftp = fopen(SU_FILE, "w");
        if (!suftp) {
            printf("Can't create file\n");
            fclose(suftp);
            return -1;
        }
    }
    system("chmod 777 /tmp/sucount"); //change this later
    putc(1, suftp);
    fclose(suftp);
}
else {
    count = getc(suftp);
    fclose(suftp);
    if (count == EOF || count < 0) {
        printf("Invalid file\n");
        return -1;
    }
    if (count >= 3) {
        //exeve(command, argv, envp);
        return 3;
    }
    else {
        count++;
        suftp = fopen(SU_FILE, "w");
        if (!suftp) {
            printf("Can't open for writing\n");
            fclose(suftp);
            return -1;
        }
        putc(count, suftp);
        fclose(suftp);
    }
}

```



```

    }
    return 1;
}
/****
 * Resets the "su" count to 0.*/
void reset_count() {
    FILE *fp;
    fp = fopen(SU_FILE, "w");
    if (fp) {
        putc(0, fp);
    }
    fclose(fp);
}
// # File wrapAC.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscalls.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include "wrapper.h"
#include "AC.c"
#include "log.c"
/* Argv: arguments to the original call & the original filename
   Env: environment strings from the original call */
int main(int argc, char *argv[], char *envp[])
{
    char * filename;
    // Strip the original filename from the args array
    argc -= 1;
    filename = argv[argc];
    argv[argc] = NULL;
    // Wrapping the wrapper could lead to an infinite loop
    if (! strcmp(filename, WRAPPER, strlen(WRAPPER))) {printf("Wrapping
error\n");}
    else {int err;
        depth = 1;
        log(LOG_FILE, filename, argc, argv);
        if (1 == AAllowed(filename))
            (err = execve(filename, argv, envp));
        else {exit(1);}
        perror("DTC");
        fprintf(stderr, "error: %d,%d\n", err, errno);
        fflush(stderr);fflush(stdout);
        return(-errno);
    }
    return -1;
}
// # File wrapBrain.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscalls.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <signal.h>
#include "wrapper.h"
#define _MSG_SIZE_ 255
struct msgbuf
{
    int mtype;
    int32_t opcode;
    char mtext[_MSG_SIZE_];
};
int32_t qid=-1;
struct msgbuf qmsg;
/****
 * Argv: arguments to the original call & the original filename
 * Env: environment strings from the original call */
int main(int argc, char *argv[], char *envp[])
{
    char * filename;
    // Strip the original filename from the args array
    argc -= 1;
    filename = argv[argc];
    argv[argc] = NULL;
    // Wrapping the wrapper could lead to an infinite loop
    if (! strcmp(filename, WRAPPER, strlen(WRAPPER)))
        (printf("Wrapping error\n");)
    else
    {
        int err;
        qid=msgget((key_t) (12345), 0);
        if(qid==-1) return(-1);
        qmsg.mtype=1;
        sprintf(qmsg.mtext, "%d %s", getpid(), filename);
        qmsg.opcode=getpid();
        if(msgsnd(qid, &qmsg, sizeof(struct msgbuf), IPC_NOWAIT)==-1)
            (perror("bad send");return -1;)
        msgrcv(qid, &qmsg, sizeof(struct msgbuf), getpid(), 0);
        if (qmsg.opcode == 0) {
            //execute the program
            err = execve(filename, argv, envp);
        } else if ((qmsg.opcode < 0) && (qmsg.opcode >= -255)) {
            //print string
            fprintf(stderr, "%s\n", qmsg.mtext);
            return qmsg.opcode;
        } else if (qmsg.opcode<-256) {
            //bounce to another server, and run the program there
            char s[1024];
            sprintf(s, "cd ");
            getcwd(s+3, 1020);
            strcat(s, " ");
            for(qid=0;qid<argc;qid++)
                (strcat(s, " ");strcat(s, argv[qid]));
            newargv[0]="ssh";
            newargv[1]=qmsg.mtext;
            newargv[2]=s;
        }
    }
}

```

```

newargv[3]=NULL;
err = execve("/usr/local/bin/ssh", newargv, envp);
} else if (qmsg.opcode == 255) {
    //kill, kill, KILL!!!
    kill(getpid(), SIGTERM);
}
perror("Exec Failed");
fprintf(stderr, "error: %d,%d\n", err, errno);
fflush(stderr);
fflush(stdout);
printf("exiting:%d\n", -errno);
return(-errno);
}
printf("Houston, we have a problem!\n");
return -1;
}
// # File wrapd.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscalls.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include "wrapper.h"
#include "PL.c"
#include "D.c"
/* Argv: arguments to the original call & the original filename */
int main(int argc, char *argv[], char *envp[])
{char *filename;
FILE *logfile;
// Strip the original filename from the args array
argc -= 1; filename = argv[argc]; argv[argc] = NULL;
// Wrapping the wrapper could lead to an infinite loop
if (! strcmp(filename, WRAPPER, strlen(WRAPPER))) {printf("Wrapping
error!\n");}
else {int err;
err = plog(filename, argc, argv);
switch (err)
{case 1: case 0: (if (Dallowed(pathname) == 1) err =
execve(filename, argv, envp);
LOGIT("-X-");perror("Exec failed");
fprintf(stderr, "error: %d,%d %s\n", err, errno,
filename);
fflush(stderr);fflush(stdout);return(-errno);}
case FORKERR: (LOGIT("-F-");printf("%s\n",
FORKERRMSG);exit(1);}
case FDERR: (LOGIT("-D-");printf("%s\n", FDERRMSG);exit(1);}
case PIPEERR: (LOGIT("-P-");printf("%s\n",
PIPEERRMSG);exit(1);}
case WAITERR: (LOGIT("-S-");printf("%s\n",
WAITERRMSG);exit(1);}
case GETWDERR: (LOGIT("-W-");printf("%s\n",
GETWDERRRMSG);exit(1);}
case NOLOGERR: (LOGIT("-L-");if (getpid() < SYSPROCTHRESH) (err
= execve(filename,
argv, envp);}
printf("%s\n", NOLOGERRMSG);exit(1);}
case NOMEMERR: (LOGIT("-M-");printf("%s\n",
NOMEMERRMSG);exit(1);}
default: (LOGIT("-O-");printf("Process Error %d\n",
err);perror("Ouch!!!");exit(1);}
}
}
return -1;
}
// # File wrapipc.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscalls.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "wrapper.h"
#define _MSG_SIZE_ 255
struct msgbuf
{
    int mtype;
    int32_t opcode;
    char mtext[_MSG_SIZE_];
};
int32_t qid=-1;
//memcpy(qmsg.mtext, "Hello World", _MSG_SIZE_);
sprintf(qmsg.mtext, "%d %s", getpid(), filename);
qmsg.opcode=getpid();
//printf("Sending");
if(msgsnd(qid, &qmsg, sizeof(struct msgbuf), IPC_NOWAIT)==-1)
{perror("bad send");return -1;}
//perror("bad send");return(-1);}
//printf("Receiving");
msgrcv(qid, &qmsg, sizeof(struct msgbuf), getpid(), 0);
printf("%d %d %s", qmsg.mtype, qmsg.opcode, qmsg.mtext);
if(qmsg.opcode<0)
{
    fprintf(stderr, "%s\n", qmsg.mtext);
    return -qmsg.opcode;
}
else if(qmsg.opcode>0)
{
    char* newargv[1024];
    char s[1024];
    switch(qmsg.opcode)
    {
        case 1 :
            //--help
            break;
        case 2 :
            //--help
            break;
    }
}

```

```

case 3 :
//ssh
    sprintf(s, "cd ");
    getcwd(s+3, 1020);
    strcat(s, " ");
    for(qid=0;qid<argc;qid++)
        (strcat(s, " ");strcat(s, argv[qid]));
    newargv[0]="ssh";
    //newargv[1]="10.0.0.123";
    newargv[1]=qmsg.mtext;
    newargv[2]=s;
    newargv[3]=NULL;
    err = execve("/usr/local/bin/ssh", newargv, envp);
break;
case 4 :
//blablablah
break;
//...
case 42 :
    err = execve(filename, argv, envp);
break;
default :
    return -1;
}
}
else
    err = execve(filename, argv, envp);
//printf("Executing");
perror("Exec Failed");
fprintf(stderr, "error: %d,%d\n", err, errno);
fflush(stderr);
fflush(stdout);
printf("exiting:%d\n", -errno);
return(-errno);
}
printf("Houston, we have a problem!\n");
return -1;
}
// # File wraplog.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscalls.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include "wrapper.h"
#include "log.c"
/* Argv: arguments to the original call & the original filename */
int main(int argc, char *argv[], char *envp[])
{char * filename;
  argc -= 1;filename = argv[argc];argv[argc] = NULL; // Strip the
  original filename from
  the args array
  // Wrapping the wrapper could lead to an infinite loop

```

```

if (! strcmp(filename, WRAPPER, strlen(WRAPPER))) {printf("Wrapping
error\n");}
else {int err;
  log(LOG_FILE, filename, argc, argv);
  err = execve(filename, argv, envp);
  perror("Exec failed");
  fprintf(stderr, "error: %d,%d\n", err, errno);
  fflush(stderr);fflush(stdout);
  printf("exiting:%d\n", -errno);
  return(-errno);
}
return -1;
}
// # File wrapNPS.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscalls.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include "wrapper.h"
#include "AC.c"
#include "log.c"
/* Argv: arguments to the original call & the original filename
  Env: environment strings from the original call */
int main(int argc, char *argv[], char *envp[])
{char * filename;
  // Strip the original filename from the args array
  argc -= 1;
  filename = argv[argc];
  argv[argc] = NULL;
  // Wrapping the wrapper could lead to an infinite loop
  if (! strcmp(filename, WRAPPER, strlen(WRAPPER))) {printf("Wrapping
  error\n");}
  else {int err;
    depth = 1;
    log(LOG_FILE, filename, argc, argv);
    if ((depth <= DEPTH_THRESH) && (1 == AAllowed(filename)))
        (err = execve(filename, argv, envp));
    else {exit(1);}
    perror("DIT");
    fprintf(stderr, "error: %d,%d\n", err, errno);
    fflush(stderr);fflush(stdout);
    return(-errno);
  }
  return -1;
}
// # File wrapnull.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscalls.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>

```

```

#include "wrapper.h"
/* Argv: arguments to the original call & the original filename */
int main(int argc, char *argv[], char *envp[])
{
    char * filename;
    // Strip the original filename from the args array
    argc -= 1; filename = argv[argc]; argv[argc] = NULL;
    // Wrapping the wrapper could lead to an infinite loop
    if (! strcmp(filename, WRAPPER, strlen(WRAPPER))) { printf("Wrapping
error\n"); exit(0); }
    else {
        int err;
        err = execve(filename, argv, envp);
        perror("Exec failed");
        // fprintf(stderr, "error: %d, %d\n", err, errno);
        fflush(stderr); fflush(stdout);
        // printf("Exiting: %d\n", -errno); return(-errno);
    }
    exit(1);
}

// # File wrapper.h
#define WRAPPER "/root/wrap"
#define LOG_FILE "/root/wraplog"
#define VERBOSE 1
#define DEBUG 1
#define SYSPROCTRESH 100
/* system process ID threshold */
// For tracking process lineage
#define PROC "/proc/"
#define STATUS "/status"
#define MAXPIDDIGITS 5
#define MAXLINELEN 500
#define MAXINPOLEN 100
#define MAXLABELLEN 10
#define MAXASCII 2048
#define PPID "PPid"
#define NAME "Name"
#define ZERO "0"
#define FORKERR -24
#define FORKERRMSG "Fork failed"
#define PDERR -25
#define PDERRMSG "Invalid file descriptor"
#define PIPEERR -26
#define PIPEERRMSG "Pipe failed"
#define WAITERR -27
#define WAITERRMSG "Process zombie"
#define GETWDERMSG "Invalid working directory"
#define NOLOGERR -29
#define NOLOGERRMSG "No log file"
#define NOMEMERR -30
#define NOMEMERRMSG "Out of memory"
// For the su demo
#define DEMO 0 /* If DEMO = 1, run the su demo */
#define SU_FILE "/tmp/sucount"
#define FAKE_SU "/root/su"
/* Logging functions */

int log(char *logfile, char *program, int argc, char *argv[]); /*
Direct to file
logging */
int pilog(char *program, int argc, char *argv[]); /* process lineage
logline generation
*/
int needpathname(char *path); /* Do I need a pathname? */
void translate(char *stat); /* Change all characters to
printable representations
*/
int log_lineage(char *pid); /* short form lineage logging */
int log_lineage(FILE *filep, char *pid); /* long form lineage
logging direct to file
for log.c only */
void getinfo(char *procline, char *infobuf);
int getlabel(char *procline, char *infobuf);
void ltoa(int n, char s[]);
void reverse(char s[]);
/* Functions for a dead demo */
int process_su(char *command, char *argv[], char *envp[]);
void reset_count();
// # File wrapPLACD.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <systcall.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include "wrapper.h"
#include "PLAC.c"
#include "Pl.c"
/* Argv: arguments to the original call & the original filename */
int main(int argc, char *argv[], char *envp[])
{
    FILE *logfp;
    char * filename; // Strip the original filename from the args array
    argc -= 1; filename = argv[argc]; argv[argc] = NULL;
    // Wrapping the wrapper could lead to an infinite loop
    if (! strcmp(filename, WRAPPER, strlen(WRAPPER))) { printf("Wrapping
error\n"); }
    else {
        int err;
        err = pilog(filename, argv, argv);
        switch (err)
        {
            case 1: case 0: { if ((1 == Pallowed(lineage)) || (1 ==
Aallowed(filename))) &&
(depth < DEPTH_THRESH))
                { LOGIT("");
                    err = execve(filename, argv, envp);
                    LOGIT("-X-");
                    perror("Exec failed");
                    // fprintf(stderr, "error: %d, %d\n", err, errno,
filename);
                    fflush(stderr); fflush(stdout); return(-errno); }
                else { LOGIT("-A-"); fprintf(stderr, "%s\n",
FORKERRMSG); exit(-1); }

```

```

    }
    case FORKERR: (LOGIT("-F-");printf("%s\n",
        FORKERRMSG);exit(1);)
    case FDERR: (LOGIT("-D-");printf("%s\n", FDERRMSG);exit(1);)
    case PIPEERR: (LOGIT("-P-");printf("%s\n",
        PIPEERRMSG);exit(1);)
    case WAITERR: (LOGIT("-S-");printf("%s\n",
        WAITERRMSG);exit(1);)
    case GETWDERR: (LOGIT("-W-");printf("%s\n",
        GETWDERRMSG);exit(1);)
    case NOLOGERR: (LOGIT("-L-");if (getpid() < SYSPROCTHRESH) {err
        = execve(filename,
        argv, envp);}
        printf("%s\n", NOLOGERRMSG);exit(1);)
    case NOMEMERR: (LOGIT("-M-");printf("%s\n",
        NOMEMERRMSG);exit(1);)
    default: (LOGIT("-O-");printf("Process Error %d\n",
        err);perror("Ouch!!!");exit(1);)
    }
}
return -1;
}

// # File wrapPl.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscall.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include "wrapper.h"
#include "PLAC.c"
#include "PL.c"
/* Argv: arguments to the original call & the original filename */
int main(int argc, char *argv[], char *envp[])
{FILE *logfp;
char * filename; // Strip the original filename from the args array
argc -= 1;filename = argv[argc];argv[argc] = NULL;
// Wrapping the wrapper could lead to an infinite loop
if (!strcmp(filename, WRAPPER, strlen(WRAPPER))) {printf("Wrapping
error\n");}
else {int err;
err = plog(filename, argc, argv);
switch (err)
{case 1: case 0: {if (1 == Plallowed(lineage))
{LOGIT("");
err = execve(filename, argv, envp);
LOGIT("-X-");
perror("Exec failed");
fprintf(stderr, "error: %d,%d '%s'\n", err, errno,
filename);
fflush(stderr);fflush(stdout);return(-errno);}
else {LOGIT("-A-");fprintf(stderr,"%s\n",
FORKERRMSG);exit(-1);}
}
}

case FORKERR: (LOGIT("-F-");printf("%s\n",
        FORKERRMSG);exit(1);)
    case FDERR: (LOGIT("-D-");printf("%s\n", FDERRMSG);exit(1);)
    case PIPEERR: (LOGIT("-P-");printf("%s\n",
        PIPEERRMSG);exit(1);)
    case WAITERR: (LOGIT("-S-");printf("%s\n",
        WAITERRMSG);exit(1);)
    case GETWDERR: (LOGIT("-W-");printf("%s\n",
        GETWDERRMSG);exit(1);)
    case NOLOGERR: (LOGIT("-L-");if (getpid() < SYSPROCTHRESH) {err
        = execve(filename,
        argv, envp);}
        printf("%s\n", NOLOGERRMSG);exit(1);)
    case NOMEMERR: (LOGIT("-M-");printf("%s\n",
        NOMEMERRMSG);exit(1);)
    default: (LOGIT("-O-");printf("Process Error %d\n",
        err);perror("Ouch!!!");exit(1);)
    }
}
return -1;
}

// # File wrapPllog.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscall.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include "wrapper.h"
#include "PL.c"
/* Argv: arguments to the original call & the original filename */
int main(int argc, char *argv[], char *envp[])
{char *filename;
FILE *logfp;
// Strip the original filename from the args array
argc -= 1;filename = argv[argc];argv[argc] = NULL;
// Wrapping the wrapper could lead to an infinite loop
if (!strcmp(filename, WRAPPER, strlen(WRAPPER))) {printf("Wrapping
error\n");}
else {int err;
err = plog(filename, argc, argv);
switch (err)
{case 1: case 0: {LOGIT("");err = execve(filename, argv, envp);
LOGIT("-X-");perror("Exec failed");
fprintf(stderr, "error: %d,%d '%s'\n", err, errno,
filename);
fflush(stderr);fflush(stdout);return(-errno);}
case FORKERR: (LOGIT("-F-");fprintf(stderr, "%s\n",
        FORKERRMSG);exit(1);)
    case FDERR: (LOGIT("-D-");fprintf(stderr, "%s\n",
        FDERRMSG);exit(1);)
    case PIPEERR: (LOGIT("-P-");fprintf(stderr, "%s\n",
        PIPEERRMSG);exit(1);)
    case WAITERR: (LOGIT("-S-");fprintf(stderr, "%s\n",
        WAITERRMSG);exit(1);)
}
}
}

```

```

        case GETWDERR: {LOGIT("-W-"); fprintf(stderr, "%s\n",
GETWDERMSG); exit(1);}
        case NOLOGERR: {LOGIT("-L-"); err = execve(filename, argv, envp);
            printf("%s\n", NOLOGERRMSG); exit(1);}
        case NOMEMERR: {LOGIT("-M-"); fprintf(stderr, "%s\n",
NOMEMERRMSG); exit(1);}
        default: {LOGIT("-O-"); fprintf(stderr, "Process Error %d\n",
err); perror("Ouch!!!"); exit(1);}
    }
    return -1;
}

// # File wrapsSH.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <syscalls.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include "wrapper.h"
#include "AC.c"
#include "PL.c"
/*****
* Argv: arguments to the original call & the original filename
* Env: environment strings from the original call
*/
int main(int argc, char *argv[], char *envp[])
{char * filename;
char * newargv[1024];
int i;
// Strip the original filename from the args array
argc -= 1;
filename = argv[argc];
argv[argc] = NULL;
// Wrapping the wrapper could lead to an infinite loop
if (! strcmp(filename, WRAPPER, strlen(WRAPPER))) {printf("Wrapping
error\n");}
else if (1 == ACallowed(filename))
{int err; err = execve(filename, argv, envp);
perror("Exec failed");
fprintf(stderr, "error: %d, %d\n", err, errno);
fflush(stderr); fflush(stdout);
printf("exiting: %d\n", -errno); return(-errno);
}
else {int err;
plog(filename, argc, argv);
printf(s, "cd ");
getcwd(s+3, 1010);
strcat(s, ";");
for(i=0; i<argc; i++) {strcat(s, " "); strcat(s, argv[i]);}
newargv[0] = "/usr/local/bin/ssh";
newargv[1] = "10.0.0.123";
newargv[2] = s;
newargv[3] = NULL;

        err = execve("/usr/local/bin/ssh", newargv, envp);
    }
    perror("DTK");
    return(-errno);
}

/* Example Kernel changes
The following series of 'diff' output describes an example of code
changes made to a Linux 2.4.18 kernel
in order to implement a wrapper according to specific embodiments of
the present invention and also to
implement other operating system modifications for deception. Lines
starting with '#' diff' indicate first
the name of the new kernel file and which code lines are deleted or
added from the original kernel file.
This code could be used with a patch command or program in Linux to
automatically modify the
indicated files as will be understood to persons of skill in the art.
Program changes for wrapper include files:
exec.c, sched.h, process.c, entry.S
Program changes for the module include:
ksyms.c, user.c, dnortify.c
# diff exec.c /usr/src/linux-2.4.18/fs/exec.c
45,53d44
< #include <asm/ptrace.h> // need to access pt_regs fields
< #include <linux/sched.h> // need WRAP_FLAG, WRAP_PATH
< #include <linux/elf.h> // For rec_elf_binary()
< #define DEBUGGING 0 // print debugging statements
< #define DEBUG_MEM 0
58,77d48
< // For rec_misc_binary() (taken from binfmt_misc.c)
< typedef struct {
< struct list_head list;
< int flags; // type, status, etc. */
< int offset; // offset of magic */
< int size; // size of magic/mask */
< char *magic; // magic or filename extension */
< char *mask; // mask, NULL for exact match */
< char *interpreter; // filename of interpreter */
< char *name;
< struct dentry *dentry;
< } Node;
< static rwlock_t entries_lock __attribute__((unused)) =
RW_LOCK_UNLOCKED;
< enum {Enabled, Magic};
< static LIST_HEAD(entries);
83,114d53
< /*sys_ad -- checks if a certain pid is the actual wrapper process.
< the wrapper process has a flag set in the flags field of the
task_struct.
< the flag is defined as WRAP_FLAG in linux/sched.h.
< */
< /** DTK(dcm)--- start **/
< asmlinkage long sys_ad(int pid) {
< struct task_struct *p;
< if(current->uid) { //if not root, return original error

```

```

< //printr(KERN_CRIT "sys_ad1\n");
< return -ENOSYS;
> }
< p=find_task_by_pid(pid);
< if(p=NULL) { //if no such pid, return standard error
< //printr(KERN_CRIT "sys_ad2\n");
< return -ENOSYS;
> }
< if(p->flags&WRAP_FLAG) { //if it's the wrapper return different
error
< //printr(KERN_CRIT "sys_ad3\n");
< return -EAGAIN;
> }
< //printr(KERN_CRIT "sys_ad4\n");
< return -ENOSYS; // otherwise return original error
> }
< /** DTK(dcm)--- end **/
221a161
226c166
< if (get_user(p, argv)) {
---
< if (get_user(p, argv))
228,229c168
> }
< if (!p) {
---
< if (!p)
231d169
233c171
< if(++i > max) {
---
< if(++i > max)
235d172
> }
< 253,254c190
< if (get_user(str, argv+argc) || !len = strlen_user(str, bprm-
>p)) {
< printr(KERN_CRIT "copy_strings 1\n");
---
< if (get_user(str, argv+argc) || !len = strlen_user(str, bprm-
>p)) {
256d191
279d213
< if (DEBUG_MEM) printr("page allocated from copy_strings
(&s)\n", bprm->file-
>f_dentry->d_name.name);
295,296c229
< if (err) {
< printr(KERN_CRIT "copy_strings 2\n");
---
< if (err)
298d230
414,416c346

```

```

< if (DEBUGING) printr(KERN_CRIT "open_exec(&s)\n", name);
< if (path_init(name, LOOKUP_FOLLOW|LOOKUP_POSITIVE, &nd)) {
---
> if (path_init(name, LOOKUP_FOLLOW|LOOKUP_POSITIVE, &nd))
418,419d347
421d348
427,428c354
< int err = permission(inode, MAY_EXEC);
---
> int err = permission(inode, MAY_EXEC);
432d357
434d358
< if (DEBUG_MEM) printr(KERN_CRIT "s: about to dentry_open
from open_exec\n",
name);
842d765
886c809
< continue;
---
> continue;
929,1115d851
< /* DTK: The rec_xxx_binary() methods return 1 if the binary format
is
* recognized and 0 otherwise. The checks are taken directly from
the
* load_xxx_binary() methods defined in fs/binfmt_XXX.c.
*/
< int rec_aout_binary(struct linux_binprm *bprm, struct pt_regs *regs)
{
< struct exec ex;
< ex = *((struct exec *) bprm->buf); /* exec-header */
< if ((N_MAGIC(ex) != ZMAGIC && N_MAGIC(ex) != OMAGIC &&
< N_MAGIC(ex) != QMAGIC && N_MAGIC(ex) != NMAGIC) ||
< N_TRSIZE(ex) || N_DRSIZE(ex) ||
< bprm->file->f_dentry->d_inode->i_size <
< ex.a_text+ex.a_data+N_SYMSIZE(ex)+N_TXTOFF(ex)) {
< return -ENOEXEC;
> }
< else { return 1; }
> }
< int rec_elf_binary(struct linux_binprm *bprm, struct pt_regs *regs)
{
< struct elfhdr elf_ex;
< elf_ex = *((struct elfhdr *) bprm->buf);
< if (memcmp(elf_ex.e_ident, ELF_MAGIC, SELFMAGIC) != 0) ||
< (elf_ex.e_type != ET_EXEC && elf_ex.e_type != ET_DYN) ||
< (elf_check_arch(&elf_ex)) ||
< (!bprm->file->f_op || !bprm->file->f_op->mmap) ) {
< return -ENOEXEC;
> }
< else {
< return 1;
> }
< }
< static Node *check_file(struct linux_binprm *bprm)

```

```

< {
<   char *p = strchr(bprm->filename, '.');
<   struct list_head *l;
<   for (l = entries.next; l != &entries; l = l->next) {
<     Node *e = list_entry(l, Node, list);
<     char *s;
<     int j;
<     if (!test_bit(Enabled, &e->flags))
<       continue;
<     if (!test_bit(Magic, &e->flags)) {
<       if (p && strcmp(e->magic, p + 1))
<         return e;
<       continue;
<     }
<     s = bprm->buf + e->offset;
<     if (e->mask) {
<       for (j = 0; j < e->size; j++)
<         if ((*s++ ^ e->magic[j]) & e->mask[j])
<           break;
<     } else {
<       for (j = 0; j < e->size; j++)
<         if ((*s++ ^ e->magic[j]))
<           break;
<     }
<     if (j == e->size)
<       return e;
<   }
<   return NULL;
< }
< int rec_misc_binary(struct linux_binprm *bprm, struct pt_regs *regs)
< {
<   Node *fnt;
<   char iname[BINPRM_BUF_SIZE];
<   read_lock(&entries_lock);
<   fnt = check_file(bprm);
<   if (fnt) {
<     strncpy(iname, fnt->interpreter, BINPRM_BUF_SIZE - 1);
<     iname[BINPRM_BUF_SIZE - 1] = '\0';
<   }
<   read_unlock(&entries_lock);
<   if (!fnt) { return -ENOEXEC; }
<   else { return 1; }
< }
< int rec_script_binary(struct linux_binprm *bprm, struct pt_regs
*regs) {
<   char *cp, *i_name, *i_arg;
<   struct file *file;
<   char interp[BINPRM_BUF_SIZE];
<   int retval;
<   if ((bprm->buf[0] != '#') || (bprm->buf[1] != 'i') || (bprm-
>sh_bang)) {
<     if (DEBUGGING) printk(KERN_CRIT "#! not recognized\n");
<     return -ENOEXEC;
<   }
<   // DTK: Now make sure the specified interpreter can be executed
<   bprm->sh_bang++;
<
<   bprm->buf[BINPRM_BUF_SIZE - 1] = '\0';
<   if ((cp = strchr(bprm->buf, '\n')) == NULL)
<     cp = bprm->buf + BINPRM_BUF_SIZE - 1;
<   *cp = '\0';
<   while (cp > bprm->buf) {
<     cp--;
<     if ((*cp == ' ') || (*cp == '\t'))
<       *cp = '\0';
<     else
<       break;
<   }
<   for (cp = bprm->buf+2; (*cp == ' ') || (*cp == '\t'); cp++);
<   if (*cp == '\0')
<     return -ENOEXEC; /* No interpreter name found */
<   i_name = cp;
<   i_arg = 0;
<   for (; *cp && (*cp != ' ') && (*cp != '\t'); cp++)
<     /* nothing */;
<   while ((*cp == ' ') || (*cp == '\t'))
<     *cp++ = '\0';
<   if (*cp)
<     i_arg = cp;
<   strcpy(interp, i_name);
<   if (DEBUGGING) printk(KERN_CRIT "interp:%s\n", interp);
<   /*< OK, we've parsed out the interpreter name and
<   * (optional) argument.
<   * Splice in (1) the interpreter's name for argv[0]
<   * (2) (optional) argument to interpreter
<   * (3) filename of shell script (replace argv[0])
<   * This is done in reverse order, because of how the
<   * user environment and arguments are stored.
<   */
<   remove_arg_zero(bprm);
<   retval = copy_strings_kernel(1, &bprm->filename, bprm);
<   if (retval < 0) return retval;
<   bprm->argc++;
<   if (i_arg) {
<     retval = copy_strings_kernel(1, &i_arg, bprm);
<     if (retval < 0) return retval;
<     bprm->argc++;
<   }
<   retval = copy_strings_kernel(1, &i_name, bprm);
<   if (retval < 0) return retval;
<   bprm->argc++;
<   /*< OK, now restart the process with the interpreter's dentry.
<   */
<   file = open_exec(interp);
<   if (!file)
<     printk(KERN_CRIT "file is NULL\n");
<   if (IS_ERR(file)) {
<     return PTR_ERR(file);
<   }
<   bprm->file = file;
<   retval = prepare_binprm(bprm);
<   if (retval < 0)
<     goto out;

```



Program Code Listing Appendix (Paper Duplicate of Compact Disc File Cohen\_et\_al.txt)

```

< // Specified interpreter is executable
< retval = 1;
< out:
< if (file) {
<     allow_write_access(file);
<     fput(file);
< }
< return retval;
1119,1123d854
< }
< * DTK:
< * The EXEC_CKD flag keeps track of the recursion-
< * - EXEC_CKD on: filename has been checked for exec errors and has
none
< * - EXEC_CKD off: filename has not been checked for exec errors
yet
1129c860
< int retval, retval_aout, retval_elf, retval_misc, retval_script;
---
< int retval;
1131d861
< char * orig_filename;
1133,1141d862
< if (DEBUGGING) printf(KERN_CRIT "do_execve(%) \n", filename);
< orig_filename = filename;
< if ( (current->flags & EXEC_CKD) && (current->flags & WRAP_FLAG) )
{
<     filename = WRAP_PATH;
}
1143d863
< retval = PTR_ERR(file);
1145c865,866
< if (IS_ERR(file))
---
< retval = PTR_ERR(file);
< if (IS_ERR(file))
1157,1162c878,879
< if (file) {
<     allow_write_access(file);
<     fput(file);
}
---
< allow_write_access(file);
< fput(file);
1167,1172c884,885
< if (file) {
<     allow_write_access(file);
<     fput(file);
}
---
< allow_write_access(file);
< fput(file);
1174a888
1176,1177c890,891
< if (retval < 0)
<     goto out;

```

```

---
> if (retval < 0)
>     goto out;
1188,1271c902,904
< // If this is the wrapper, add filename of user's original call to
the end of argv
< if ((current->flags & WRAP_FLAG) && (current->flags & EXEC_CKD)) {
<     char ** newargv;
<     int j;
<     newargv = (char **) kmalloc(sizeof(char *) * (bprm argc + 2),
GFP_KERNEL);
<     if (DEBUG_MEM) printf("kmalloc'ing from do_execve(%) \n",
filename);
<     if (! newargv) {
<         retval = -ENOMEM;
<         goto out;
<     }
<     for (j = 0; j < bprm argc; j++) { newargv[j] = argv[j]; }
<     newargv[j] = orig_filename;
<     newargv[j+1] = NULL;
<     bprm argc++;
<     //(char *) (*regs).ebx = filename;
<     //(char *) (*regs).ecx = newargv;
<     retval = copy_strings_kernel(bprm argc, newargv, &bprm);
<     kfree(newargv);
<     if (DEBUG_MEM) printf("kfree'd from do_execve(%) \n",
filename);
}
< }
< else {
<     retval = copy_strings(bprm argc, argv, &bprm);
}
< if (retval < 0) {
<     goto out;
}
< if ( ( (current->flags & EXEC_CKD) && (current->flags &
WRAP_FLAG) ) ) {
<     // Don't run wrapper if we don't recognize the executable format
<     retval_aout = rec_aout_binary(&bprm, regs);
<     retval_elf = rec_elf_binary(&bprm, regs);
<     retval_misc = rec_misc_binary(&bprm, regs);
<     retval_script = rec_script_binary(&bprm, regs);
<     if ((retval_aout < 0) && (retval_aout != -ENOEXEC)) {
<         retval = retval_aout;
<         goto out;
}
< if ((retval_elf < 0) && (retval_elf != -ENOEXEC)) {
<     retval = retval_elf;
<     goto out;
}
< if ((retval_misc < 0) && (retval_misc != -ENOEXEC)) {
<     retval = retval_misc;
<     goto out;
}
< if ((retval_script < 0) && (retval_script != -ENOEXEC)) {
<     retval = retval_script;
<     goto out;
}

```

```

< >
< > if ((retval_aout < 0) && (retval_elf < 0) && (retval_misc < 0)
&&
< > (retval_script < 0)) {
< > if (DEBUGGING) printf(KERN_CRIT "Executable type not
recognized (%s)\n",
filename);
< > current->flags |= EXEC_CKD;
< >
< > }
< > // DTK: No exec errors.. Now we can call the wrapper
< > if ( (! (current->flags & EXEC_CKD)) && (current->flags &
WRAP_FLAG) ) {
< > current->flags |= EXEC_CKD;
< > if (file) {
< > allow_write_access(file);
< > fput(file);
< > }
< > for (i = 0 ; i < MAX_ARG_PAGES ; i++) {
< > struct page * page = bprm.page[i];
< > if (page)
< > _free_page(page);
< > }
< > return do_execve(filename, argv, envp, regs);
< > }
< > else {
< > current->flags &= ~EXEC_CKD;
< > }
< > }
---
< > retval = copy_strings(bprm.argv, argv, &bprm);
< > if (retval < 0)
1274,1275c907
< > goto out;
< > if (retval >= 0) {
1277,1278c909
< > return retval;
< > }
---
< > return retval;
1282,1288c913,915
< > if (file) {
< > allow_write_access(file);
< > fput(file);
< > }
---
< > allow_write_access(bprm.file);
< > if (bprm.file)
< > fput(bprm.file);
1295,1297d921
< > current->flags &= ~EXEC_CKD;
# diff sched.h /usr/src/linux-2.4.18/include/linux/sched.h
32,37d31
< #define WRAP_PATH "/root/wrap" /* location of the execve wrapper
executable */

```

```

< #define WRAP_THRESH 20 /* don't run wrapper on PIDs less than this
threshold */
<
439,443d432
< #define WRAP_FLAG 0x08000000 /* Process has been wrapped */
< #define EXEC_CKD 0x04000000 /* Process has been checked for exec
errors */
453,456d441
< #define PT_HIDETRACE 0x00000020 /* conceal wrapper from execution
trace */
# diff process.c /usr/src/linux-2.4.18/arch/i386/kernel/process.c
774d773
777,784d775
< * DTK:
< * If the WRAP_FLAG is on, this is the wrapper's execve call.
< * - Turn off the wrap flag
< * - Turn execution tracing back on
< * If the WRAP_FLAG is off, this is the user's execve call.
< * - Turn on the wrap flag
< * - Turn off the execution trace for this call
795,815d785
< /**** DTK-- ***/
< if (current->pid < WRAP_THRESH) {
< /* Do nothing-- don't run wrapper on boot-up */
< }
< else if ((current->flags) & WRAP_FLAG) {
< current->flags &= ~WRAP_FLAG;
< if (current->ptrace & PT_HIDETRACE) {
< current->ptrace |= PT_HIDETRACE;
< current->ptrace &= ~PT_PTRACED;
< }
< }
817,821c787,788
< if (error == 0) {
< current->ptrace &= ~PT_DTRACE;
< }
---
< if (error == 0)
< current->ptrace &= ~PT_DTRACE;
824,833d790
< if (error != 0) {
< //printf(KERN_CRIT "ERROR:%d (%s)\n", error, filename);
< current->flags &= ~WRAP_FLAG;
< if (current->ptrace & PT_HIDETRACE) {
< current->ptrace |= PT_PTRACED;
< current->ptrace &= ~PT_HIDETRACE;
< }
< }
< /**** --DTK ***/

```

```

836d792
> diff entry.S /u/usr/src/linux-2.4.18/arch/i386/kernel/entry.S
396c399
< .long SYMBOL_NAME(sys_ad) /* 0 - old "setup()" system call */
---
> .long SYMBOL_NAME(sys_ni_syscall) /* 0 - old "setup()" system
call */
# diff ksyms.c /u/usr/src/linux-2.4.18/kernel/ksyms.c
75,103d74
-----dcvn-----
< //needed for the EXPORT_SYMBOL declarations
< #define UIDHASH_BITS 8
< #define UIDHASH_SZ (1 << UIDHASH_BITS)
< extern kmem_cache_t *uid_cache;
< extern struct user_struct *uidhash_table[UIDHASH_SZ];
< extern spinlock_t uidhash_lock;
< extern kmem_cache_t *dn_cache;
< extern rwlock_t dn_lock;
< extern int sock_map_fd(struct socket *sock);
< extern struct module *module_list;
< //-----dcvn-----
< /* extra symbols needed for kernel module */
< //dcvn
< EXPORT_SYMBOL(uid_cache); //for setuid sys_calls
< EXPORT_SYMBOL(uidhash_lock);
< EXPORT_SYMBOL(uidhash_table);
< EXPORT_SYMBOL(dn_lock); //for write() sys_call
< EXPORT_SYMBOL(dn_cache);
< EXPORT_SYMBOL(do_truncate);
< EXPORT_SYMBOL(sock_map_fd);
< EXPORT_SYMBOL(module_list);
< /* original code from here on */
# diff user.c /u/usr/src/linux-2.4.18/kernel/user.c
25,27c25,27
< kmem_cache_t *uid_cache;
< struct user_struct *uidhash_table[UIDHASH_SZ];
< spinlock_t uidhash_lock = SPIN_LOCK_UNLOCKED;
---
> static kmem_cache_t *uid_cache;
> static struct user_struct *uidhash_table[UIDHASH_SZ];
> static spinlock_t uidhash_lock = SPIN_LOCK_UNLOCKED;
# diff dmofity.c /u/usr/src/linux-2.4.18/fs/dmofity.c
27,28c27,28
< rwlock_t dn_lock = RW_LOCK_UNLOCKED;
< kmem_cache_t *dn_cache;
---
> static rwlock_t dn_lock = RW_LOCK_UNLOCKED;
> static kmem_cache_t *dn_cache;
# module.c
# include <linux/config.h>
# include <linux/mm.h>
# include <linux/module.h>
# include <asm/module.h>
# include <asm/uaccess.h>
# include <linux/vmalloc.h>
# include <linux/smp_lock.h>

```

```

#include <asm/pgalloc.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/kmod.h>
#include <linux/seq_file.h>
#if defined(CONFIG_MODULES) || defined(CONFIG_KALLSYMS)
extern struct module_symbol __start__ksymtab[];
extern struct module_symbol __stop__ksymtab[];
extern const struct exception_table_entry __start__ex_table[];
extern const struct exception_table_entry __stop__ex_table[];
extern const char __start__kallsyms[] __attribute__((weak));
extern const char __stop__kallsyms[] __attribute__((weak));
struct module kernel_module =
{
    size_of_struct: sizeof(struct module),
    name: "",
    uc: (ATOMIC_INIT(1)),
    flags: MOD_RUNNING,
    syms: __start__ksymtab,
    ex_table_start: __start__ex_table,
    ex_table_end: __stop__ex_table,
    kallsyms_start: __start__kallsyms,
    kallsyms_end: __stop__kallsyms,
};
struct module *module_list = &kernel_module;
#endif /* defined(CONFIG_MODULES) || defined(CONFIG_KALLSYMS) */
/* inter_module functions are always available, even when the kernel
is
* compiled without modules. Consumers of inter_module_xxx routines
* will always work, even when both are built into the kernel, this
* approach removes lots of #ifdefs in mainline code. */
static struct list_head time_list = LIST_HEAD_INIT(time_list);
static spinlock_t time_lock = SPIN_LOCK_UNLOCKED;
static int kmallocc_failed;
/* This lock prevents modifications that might race the kernel fault
* fixups. It does not prevent reader walks that the modules code
* does. The kernel lock does that.
* Since vmalloc fault fixups occur in any context this lock is taken
* irqsavely at all times. */
spinlock_t modlist_lock = SPIN_LOCK_UNLOCKED;
/* inter_module_register - register a new set of inter module data.
* @im_name: an arbitrary string to identify the data, must be unique
* @userdata: pointer to arbitrary userdata to be registered
* Description: Check that the im_name has not already been
registered.
* complain if it has. For new data, add it to the inter_module_entry
* list. */
void inter_module_register(const char *im_name, struct module *owner,
const void
*userdata)
{
    struct list_head *tmp;
    struct inter_module_entry *ime, *ime_new;
    if (!ime_new = kmalloc(sizeof(*ime), GFP_KERNEL)) {
        /* Overloaded kernel, not fatal */

```

```

        printk(KERN_ERR
               "Aieee, inter_module_unregister: cannot kcalloc entry for
               '%s'\n",
               im_name);
        kcalloc_failed = 1;
        return;
    }
    memset(&ime_new, 0, sizeof(*ime_new));
    ime_new->im_name = im_name;
    ime_new->owner = owner;
    ime_new->userdata = userdata;
    spin_lock(&ime_lock);
    list_for_each(tmp, &ime_list) {
        ime = list_entry(tmp, struct inter_module_entry, list);
        if (strcmp(ime->im_name, im_name) == 0) {
            spin_unlock(&ime_lock);
            kfree(ime_new);
            /* Program logic error, fatal */
            printk(KERN_ERR "inter_module_unregister: duplicate im_name
            BUG();
        }
    }
    list_add(&(ime_new->list), &ime_list);
    spin_unlock(&ime_lock);
}

/* inter_module_unregister - unregister a set of inter module data.
 * @im_name: an arbitrary string to identify the data, must be unique
 * Description: Check that the im_name has been registered, complain
 * if it has not. For existing data, remove it from the
 * inter_module_entry list. */
void inter_module_unregister(const char *im_name)
{
    struct list_head *tmp;
    struct inter_module_entry *ime;
    spin_lock(&ime_lock);
    list_for_each(tmp, &ime_list) {
        ime = list_entry(tmp, struct inter_module_entry, list);
        if (strcmp(ime->im_name, im_name) == 0) {
            list_del(&(ime->list));
            spin_unlock(&ime_lock);
            kfree(ime);
            return;
        }
    }
    spin_unlock(&ime_lock);
    if (kcalloc_failed) {
        printk(KERN_ERR
               "inter_module_unregister: no entry for '%s', "
               "probably caused by previous kcalloc failure\n",
               im_name);
        return;
    }
    else {
        /* Program logic error, fatal */
    }
}

        printk(KERN_ERR "inter_module_unregister: no entry for '%s'",
        im_name);
        BUG();
    }
}
/**
 * inter_module_get - return arbitrary userdata from another module.
 * @im_name: an arbitrary string to identify the data, must be unique
 * Description: If the im_name has not been registered, return NULL.
 * Try to increment the use count on the owning module, if that fails
 * then return NULL. Otherwise return the userdata.*/
const void *inter_module_get(const char *im_name)
{
    struct list_head *tmp;
    struct inter_module_entry *ime;
    const void *result = NULL;
    spin_lock(&ime_lock);
    list_for_each(tmp, &ime_list) {
        ime = list_entry(tmp, struct inter_module_entry, list);
        if (strcmp(ime->im_name, im_name) == 0) {
            if (try_inc_mod_count(ime->owner))
                result = ime->userdata;
            break;
        }
    }
    spin_unlock(&ime_lock);
    return(result);
}
/**
 * inter_module_get_request - im get with automatic request_module.
 * @im_name: an arbitrary string to identify the data, must be unique
 * @modname: module that is expected to register im_name
 * Description: If inter_module_get fails, do request_module then
 * retry.*/
const void *inter_module_get_request(const char *im_name, const char
*modname)
{
    const void *result = inter_module_get(im_name);
    if (!result) {
        request_module(modname);
        result = inter_module_get(im_name);
    }
    return(result);
}
/* * inter_module_put - release use of data from another module.
 * @im_name: an arbitrary string to identify the data, must be unique
 * Description: If the im_name has not been registered, complain,
 * otherwise decrement the use count on the owning module.*/
void inter_module_put(const char *im_name)
{
    struct list_head *tmp;
    struct inter_module_entry *ime;
    spin_lock(&ime_lock);
    list_for_each(tmp, &ime_list) {
        ime = list_entry(tmp, struct inter_module_entry, list);

```

```

        if (strcmp(ime->im_name, im_name) == 0) {
            if (ime->owner)
                __MOD_DEC_USE_COUNT(ime->owner);
            spin_unlock(&ime_lock);
            return;
        }
        spin_unlock(&ime_lock);
        printk(KERN_ERR "inter_module_put: no entry for '%s', im_name);
        BUG();
    }

    #if defined(CONFIG_MODULES) /* The rest of the source */
    static long get_mod_name(const char *user_name, char **buf);
    static void put_mod_name(char *buf);
    struct module *find_module(const char *name);
    void free_module(struct module *, int tag_freed);
    /* Called at boot time */
    void __init init_modules(void)
    {
        kernel_module_nsyms = __stop__ksymtab - __start__ksymtab;
        arch_init_modules(&kernel_module);
    }

    /* Copy the name of a module from user space. */
    static inline long
    get_mod_name(const char *user_name, char **buf)
    {
        unsigned long page;
        long retval;
        page = __get_free_page(GFP_KERNEL);
        if (!page)
            return -ENOMEM;
        retval = strncpy_from_user((char *)page, user_name, PAGE_SIZE);
        if (retval > 0) {
            if (retval < PAGE_SIZE) {
                *buf = (char *)page;
                return retval;
            }
            retval = -ENAMETOOLONG;
        } else if (!retval)
            retval = -EINVAL;
        free_page(page);
        return retval;
    }

    static inline void
    put_mod_name(char *buf)
    {
        free_page((unsigned long)buf);
    }

    /* Allocate space for a module. */
    asmlinkage unsigned long
    sys_create_module(const char *name_user, size_t size)
    {
        char *name;
        long namelen, error;
        struct module *mod;
        unsigned long flags;

        if (!capable(CAP_SYS_MODULE))
            return -EPERM;
        lock_kernel();
        if ((namelen = get_mod_name(name_user, &name)) < 0) {
            error = namelen;
            goto err0;
        }
        if ((mod = find_module(name)) != NULL) {
            error = -EXIST;
            goto err1;
        }
        if ((mod = (struct module *)module_map(size)) == NULL) {
            error = -ENOMEM;
            goto err1;
        }
        memset(mod, 0, sizeof(*mod));
        mod->size_of_struct = sizeof(*mod);
        mod->name = (char *) (mod + 1);
        mod->size = size;
        memcpy((char *) (mod + 1), name, namelen + 1);
        put_mod_name(name);
        spin_lock_irqsave(&modlist_lock, flags);
        mod->next = module_list;
        module_list = mod; /* link it in */
        spin_unlock_irqrestore(&modlist_lock, flags);
        error = (long) mod;
        goto err0;
    }

    err1:
        put_mod_name(name);
    err0:
        unlock_kernel();
        return error;
    }

    /* Initialize a module. */
    asmlinkage long
    sys_init_module(const char *name_user, struct module *mod_user)
    {
        struct module mod_tmp, *mod;
        char *name, *n_name, *name_tmp = NULL;
        long namelen, n_namelen, i, error;
        unsigned long mod_user_size;
        struct module_ref *dep;
        if (!capable(CAP_SYS_MODULE))
            return -EPERM;
        lock_kernel();
        if ((namelen = get_mod_name(name_user, &name)) < 0) {
            error = namelen;
            goto err0;
        }
        if ((mod = find_module(name)) == NULL) {
            error = -ENOENT;
            goto err1;
        }
    }

```

```

    )
    if ((error = get_user(mod_user_size, &mod_user->size_of_struct)) !=
        0)
        goto err1;
    if (mod_user_size < (unsigned long)&((struct module *)0L)-
        >persist_start
        || mod_user_size > sizeof(struct module) + 16*sizeof(void*)) {
        printk(KERN_ERR "init_module: Invalid module header size.\n"
            KERN_ERR "A new version of the modutils is likely "
            "needed.\n");
        error = -EINVAL;
        goto err1;
    }

    /* Hold the current contents while we play with the user's idea
       of righteouness. */
    mod_tmp = *mod;
    name_tmp = kmalloc(strlen(mod->name) + 1, GFP_KERNEL); /* Where's
        kstrdup()? */
    if (name_tmp == NULL) {
        error = -ENOMEM;
        goto err1;
    }
    strcpy(name_tmp, mod->name);
    error = copy_from_user(mod, mod_user, mod_user_size);
    if (error) {
        error = -EFAULT;
        goto err2;
    }

    /* Sanity check the size of the module. */
    error = -EINVAL;
    if (mod->size > mod_tmp.size) {
        printk(KERN_ERR "init_module: Size of initialized module "
            "exceeds size of created module.\n");
        goto err2;
    }

    /* Make sure all interesting pointers are sane. */
    if (!mod_bound(mod->name, name_len, mod)) {
        printk(KERN_ERR "init_module: mod->name out of bounds.\n");
        goto err2;
    }

    if (mod->nsyms && !mod_bound(mod->syms, mod->nsyms, mod)) {
        printk(KERN_ERR "init_module: mod->syms out of bounds.\n");
        goto err2;
    }

    if (mod->ndeps && !mod_bound(mod->deps, mod->ndeps, mod)) {
        printk(KERN_ERR "init_module: mod->deps out of bounds.\n");
        goto err2;
    }

    if (mod->init && !mod_bound(mod->init, 0, mod)) {
        printk(KERN_ERR "init_module: mod->init out of bounds.\n");
        goto err2;
    }

    if (mod->cleanup && !mod_bound(mod->cleanup, 0, mod)) {
        printk(KERN_ERR "init_module: mod->cleanup out of bounds.\n");
        goto err2;
    }

    if (mod->ex_table_start > mod->ex_table_end
        || (mod->ex_table_start &&
            !((unsigned long)mod->ex_table_start >= ((unsigned long)mod +
            mod->size_of_struct)
            && ((unsigned long)mod->ex_table_end
                < ((unsigned long)mod + mod->size)))
        || ((unsigned long)mod->ex_table_start
            - (unsigned long)mod->ex_table_end
            & sizeof(struct exception_table_entry)) {
        printk(KERN_ERR "init_module: mod->ex_table_* invalid.\n");
        goto err2;
    }

    if (mod->flags & ~MOD_AUTOCLEAN) {
        printk(KERN_ERR "init_module: mod->flags invalid.\n");
        goto err2;
    }

    if (mod_member_present(mod, can_unload)
        && mod->can_unload && !mod_bound(mod->can_unload, 0, mod)) {
        printk(KERN_ERR "init_module: mod->can_unload out of
            bounds.\n");
        goto err2;
    }

    if (mod_member_present(mod, kallsyms_end) {
        if (mod->kallsyms_end &&
            !mod_bound(mod->kallsyms_start, 0, mod) ||
            !mod_bound(mod->kallsyms_end, 0, mod)) {
            printk(KERN_ERR "init_module: mod->kallsyms out of bounds.\n");
            goto err2;
        }
        if (mod->kallsyms_start > mod->kallsyms_end) {
            printk(KERN_ERR "init_module: mod->kallsyms invalid.\n");
            goto err2;
        }
    }

    if (mod_member_present(mod, archdata_end) {
        if (mod->archdata_end &&
            !mod_bound(mod->archdata_start, 0, mod) ||
            !mod_bound(mod->archdata_end, 0, mod)) {
            printk(KERN_ERR "init_module: mod->archdata out of bounds.\n");
            goto err2;
        }
        if (mod->archdata_start > mod->archdata_end) {
            printk(KERN_ERR "init_module: mod->archdata invalid.\n");
            goto err2;
        }
    }

    if (mod_member_present(mod, kernel_data) && mod->kernel_data {
        printk(KERN_ERR "init_module: mod->kernel_data must be
            zero.\n");
        goto err2;
    }

    /* Check that the user isn't doing something silly with the name.
       */
    if ((!name_len = get_mod_name(mod->name - (unsigned long)mod
        + (unsigned long)mod_user,
        &name)) < 0) {

```

```

        printk(KERN_ERR "init_module: get_mod_name failure.\n");
        error = n_namealen;
        goto err2;
    }
    if (namealen != n_namealen || strcmp(n_name, mod_tmp.name) != 0) {
        printk(KERN_ERR "init_module: changed module name to "
            "%s" from "%s\n",
            n_name, mod_tmp.name);
        goto err3;
    }
    /* Ok, that's about all the sanity we can stomach; copy the rest.
    */
    if (copy_from_user((char *)mod+mod_user_size,
        (char *)mod_user+mod_user_size,
        mod->size-mod_user_size)) {
        error = -EFAULT;
        goto err3;
    }
    if (module_arch_init(mod))
        goto err3;
    /* On some machines it is necessary to do something here
    to make the I and D caches consistent. */
    flush_icache_range((unsigned long)mod, (unsigned long)mod + mod-
        >size);
    mod->next = mod_tmp.next;
    mod->refs = NULL;
    /* Sanity check the module's dependents */
    for (i = 0, dep = mod->deps; i < mod->ndeps; ++i, ++dep) {
        struct module *o, *d = dep->dep;
        /* Make sure the indicated dependencies are really modules. */
        if (d == mod) {
            printk(KERN_ERR "init_module: self-referential "
                "dependency in mod->deps.\n");
            goto err3;
        }
        /* Scan the current modules for this dependency */
        for (o = module_list; o != &kernel_module && o != d; o = o-
            >next)
            ;
        if (o != d) {
            printk(KERN_ERR "init_module: found dependency that is "
                "(no longer?) a module.\n");
            goto err3;
        }
    }
    /* Update module references. */
    for (i = 0, dep = mod->deps; i < mod->ndeps; ++i, ++dep) {
        struct module *d = dep->dep;
        dep->ref = mod;
        dep->next_ref = d->refs;
        d->refs = dep;
        /* Being referenced by a dependent module counts as a
        use as far as kmod is concerned. */
        d->flags |= MOD_USED_ONCE;
    }
    /* Free our temporary memory. */

    put_mod_name(n_name);
    put_mod_name(name);
    /* Initialize the module. */
    atomic_set(&mod->uc.usecount, 1);
    mod->flags |= MOD_INITIALIZING;
    if (mod->init && (error = mod->init()) != 0) {
        atomic_set(&mod->uc.usecount, 0);
        mod->flags &= ~MOD_INITIALIZING;
        if (error > 0) /* Buggy module */
            error = -EBUSY;
        goto err0;
    }
    atomic_dec(&mod->uc.usecount);
    /* And set it running. */
    mod->flags = (mod->flags | MOD_RUNNING) & ~MOD_INITIALIZING;
    error = 0;
    goto err0;
err3:
    put_mod_name(n_name);
err2:
    *mod = mod_tmp;
    strcpy((char *)mod->name, name_tmp); /* We know there is room for
    this */
err1:
    put_mod_name(name);
err0:
    unlock_kernel();
    kfree(name_tmp);
    return error;
}
spinlock_t unload_lock = SPIN_LOCK_UNLOCKED;
int try_inc_mod_count(struct module *mod)
{
    int res = 1;
    if (mod) {
        spin_lock(&unload_lock);
        if (mod->flags & MOD_DELETED)
            res = 0;
        else
            __MOD_INC_USE_COUNT(mod);
        spin_unlock(&unload_lock);
    }
    return res;
}
asmlinkage long
sys_delete_module(const char *name_user)
{
    struct module *mod, *next;
    char *name;
    long error;
    int something_changed;
    if (!capable(CAP_SYS_MODULE))
        return -EPERM;
    lock_kernel();
    if (name_user) {
        if ((error = get_mod_name(name_user, &name)) < 0)

```

```

        goto out;
        error = -ENOENT;
        if ((mod = find_module(name)) == NULL) {
            put_mod_name(name);
            goto out;
        }
        put_mod_name(name);
        error = -EBUSY;
        if (mod->refs != NULL)
            goto out;
        spin_lock(&unload_lock);
        if (!__MOD_IN_USE(mod)) {
            mod->flags |= MOD_DELETED;
            spin_unlock(&unload_lock);
            free_module(mod, 0);
            error = 0;
        } else {
            spin_unlock(&unload_lock);
            goto out;
        }
    }
    /* Do automatic reaping */
    restart:
    something_changed = 0;

    for (mod = module_list; mod != &kernel_module; mod = next) {
        next = mod->next;
        spin_lock(&unload_lock);
        if (mod->refs == NULL
            && (mod->flags & MOD_AUTOCLEAN)
            && (mod->flags & MOD_RUNNING)
            && ! (mod->flags & MOD_DELETED)
            && ! (mod->flags & MOD_USED_ONCE)
            && ! __MOD_IN_USE(mod)) {
            if ((mod->flags & MOD_VISITED)
                && ! (mod->flags & MOD_JUST_FREED)) {
                spin_unlock(&unload_lock);
                mod->flags &= ~MOD_VISITED;
            } else {
                mod->flags |= MOD_DELETED;
                spin_unlock(&unload_lock);
                free_module(mod, 1);
                something_changed = 1;
            }
        } else {
            spin_unlock(&unload_lock);
        }
    }
    if (something_changed)
        goto restart;

    for (mod = module_list; mod != &kernel_module; mod = mod->next)
        mod->flags &= ~MOD_JUST_FREED;

    error = 0;
}

out:
unlock_kernel();
return error;
}
/* Query various bits about modules. */
static int
qm_modules(char *buf, size_t bufsize, size_t *ret)
{
    struct module *mod;
    size_t nmod, space, len;
    nmod = space = 0;
    for (mod = module_list; mod != &kernel_module; mod = mod->next, ++nmod)
    {
        len = strlen(mod->name)+1;
        if (len > bufsize)
            goto calc_space_needed;
        if (copy_to_user(buf, mod->name, len))
            return -EFAULT;
        buf += len;
        bufsize -= len;
        space += len;
    }
    if (put_user(nmod, ret))
        return -EFAULT;
    else
        return -ENOSPC;
}

static int
qm_deps(struct module *mod, char *buf, size_t bufsize, size_t *ret)
{
    size_t i, space, len;
    if (mod == &kernel_module)
        return -EINVAL;
    if (!MOD_CAN_QUERY(mod))
        if (put_user(0, ret))
            return -EFAULT;
        else
            return 0;
    space = 0;
    for (i = 0; i < mod->ndeps; ++i) {
        const char *dep_name = mod->deps[i].dep->name;
        len = strlen(dep_name)+1;
        if (len > bufsize)
            goto calc_space_needed;
        if (copy_to_user(buf, dep_name, len))
            return -EFAULT;
        buf += len;
        bufsize -= len;
    }
}

```



```

        space += len;
    }
    if (put_user(i, ret))
        return -EFAULT;
    else
        return 0;
    calc_space_needed:
    space += len;
    while (++i < mod->ndeps)
        space += strlen(mod->deps[i].dep->name)+1;
    if (put_user(space, ret))
        return -EFAULT;
    else
        return -ENOSPC;
}

static int
qm_refs(struct module *mod, char *buf, size_t bufsize, size_t *ret)
{
    size_t nrefs, space, len;
    struct module_ref *ref;
    if (mod == &kernel_module)
        return -EINVAL;
    if (!MOD_CAN_QUERY(mod))
        if (put_user(0, ret))
            return -EFAULT;
        else
            return -ENOSPC;
}

static int
qm_refs(struct module *mod, char *buf, size_t bufsize, size_t *ret)
{
    space = 0;
    for (nrefs = 0, ref = mod->refs; ref ; ++nrefs, ref = ref-
>next_ref) {
        const char *ref_name = ref->ref->name;
        len = strlen(ref_name)+1;
        if (len > bufsize)
            goto calc_space_needed;
        if (copy_to_user(buf, ref_name, len))
            return -EFAULT;
        buf += len;
        bufsize -= len;
        space += len;
    }
    if (put_user(nrefs, ret))
        return -EFAULT;
    else
        return 0;
    calc_space_needed:
    space += len;
    while ((ref = ref->next_ref) != NULL)
        space += strlen(ref->ref->name)+1;
    if (put_user(space, ret))
        return -EFAULT;
    else
        return -ENOSPC;
}

static int
qm_symbols(struct module *mod, char *buf, size_t bufsize, size_t *ret)
{
    size_t i, space, len;
    struct module_symbol *s;
    char *strings;
    unsigned long *vals;
    if (!MOD_CAN_QUERY(mod))
        if (put_user(0, ret))
            return -EFAULT;
        else
            return 0;
    space = mod->nsyms * 2*sizeof(void *);
    i = len = 0;
    s = mod->syms;
    if (space > bufsize)
        goto calc_space_needed;
    if (!access_ok(VERIFY_WRITE, buf, space))
        return -EFAULT;
    bufsize -= space;
    vals = (unsigned long *)buf;
    strings = buf+space;
    for (; i < mod->nsyms ; ++i, ++s, vals += 2) {
        len = strlen(s->name)+1;
        if (len > bufsize)
            goto calc_space_needed;
        if (copy_to_user(strings, s->name, len)
            || __put_user(s->value, vals+0)
            || __put_user(space, vals+1))
            return -EFAULT;
        strings += len;
        bufsize -= len;
        space += len;
    }
    if (put_user(i, ret))
        return -EFAULT;
    else
        return 0;
    calc_space_needed:
    for (; i < mod->nsyms; ++i, ++s)
        space += strlen(s->name)+1;
    if (put_user(space, ret))
        return -EFAULT;
    else
        return -ENOSPC;
}

static int
qm_info(struct module *mod, char *buf, size_t bufsize, size_t *ret)
{
    int error = 0;
    if (mod == &kernel_module)
        return -EINVAL;
    if (sizeof(struct module_info) <= bufsize) {
        struct module_info info;
        info.addr = (unsigned long)mod;
        info.size = mod->size;
    }
}

```

```

        info.flags = mod->flags;

        /* usecount is one too high here - report appropriately to
           compensate for locking */
        info.usecount = (mod_member_present(mod, can_unload)
            && mod->can_unload ? -1 : atomic_read(&mod->uc.usecount)-
1);
    }
    if (copy_to_user(buf, &info, sizeof(struct module_info)))
        return -EFAULT;
    } else
        error = -ENOSPC;
    if (put_user(sizeof(struct module_info), ret))
        return -EFAULT;
    return error;
}

asmmlinkage long
sys_query_module(const char *name_user, int which, char *buf, size_t
bufsize,
size_t *ret)
{
    struct module *mod;
    int err;
    lock_kernel();
    if (name_user == NULL)
        mod = &kernel_module;
    else {
        long namelen;
        char *name;
        if ((namelen = get_mod_name(name_user, &name)) < 0) {
            err = namelen;
            goto out;
        }
        err = -ENOENT;
        if ((mod = find_module(name)) == NULL) {
            put_mod_name(name);
            goto out;
        }
        put_mod_name(name);
        /* __MOD_ touches the flags. We must avoid that */
        atomic_inc(&mod->uc.usecount);

        switch (which)
        {
            case 0:
                err = 0;
                break;
            case QM_MODULES:
                err = qm_modules(buf, bufsize, ret);
                break;
            case QM_DEPS:
                err = qm_deps(mod, buf, bufsize, ret);
                break;
            case QM_REFS:
                err = qm_refs(mod, buf, bufsize, ret);
                break;
            case QM_SYMBOLS:
                err = qm_symbols(mod, buf, bufsize, ret);
                break;
            case QM_INFO:
                err = qm_info(mod, buf, bufsize, ret);
                break;
            default:
                err = -EINVAL;
                break;
        }
        atomic_dec(&mod->uc.usecount);
    }

    out:
    unlock_kernel();
    return err;
}

/*Copy the kernel symbol table to user space. If the argument is
* NULL, just return the size of the table.
* This call is obsolete. New programs should use
* query_module+QM_SYMBOLS
* which does not arbitrarily limit the length of symbols.*
asmmlinkage long
sys_get_kernel_syms(struct kernel_sym *table)
{
    struct module *mod;
    int i;
    struct kernel_sym ksym;
    lock_kernel();
    for (mod = module_list, i = 0; mod; mod = mod->next) {
        /* include the count for the module name! */
        i += mod->nsyms + 1;
    }
    if (table == NULL)
        goto out;
    /* So that we don't give the user our stack content */
    memset(&ksym, 0, sizeof(ksym));
    for (mod = module_list, i = 0; mod; mod = mod->next) {
        struct module_symbol *msym;
        unsigned int j;
        if (!MOD_CAN_QUERY(mod))
            continue;
        /* magic: write module info as a pseudo symbol */
        ksym.value = (unsigned long)mod;
        ksym.name[0] = '#';
        strncpy(ksym.name+1, mod->name, sizeof(ksym.name)-1);
        ksym.name[sizeof(ksym.name)-1] = '\0';
        if (copy_to_user(table, &ksym, sizeof(ksym)) != 0)
            goto out;
        ++i, ++table;
        if (mod->nsyms == 0)
            continue;
        for (j = 0, msym = mod->syms; j < mod->nsyms; ++j, ++msym) {
            ksym.value = msym->value;

```

```

        strncpy(ksym.name, msym->name, sizeof(ksym.name));
        ksym.name[sizeof(ksym.name)-1] = '\0';
        if (copy_to_user(table, &ksym, sizeof(ksym)) != 0)
            goto out;
        ++i, ++table;
    }
    out:
    unlock_kernel();
    return i;
}

/* Look for a module by name, ignoring modules marked for deletion. */
struct module *
find_module(const char *name)
{
    struct module *mod;
    for (mod = module_list; mod; mod = mod->next) {
        if (mod->flags & MOD_DELETED)
            continue;
        if (!strcmp(mod->name, name))
            break;
    }
    return mod;
}

/* Free the given module. */
void
free_module(struct module *mod, int tag_freed)
{
    struct module_ref *dep;
    unsigned i;
    unsigned long flags;
    /* Let the module clean up. */
    if (mod->flags & MOD_RUNNING)
    {
        if (mod->cleanup)
            mod->cleanup();
        mod->flags &= ~MOD_RUNNING;
    }
    /* Remove the module from the dependency lists. */
    for (i = 0, dep = mod->deps; i < mod->ndeps; ++i, ++dep) {
        struct module_ref **pp;
        for (pp = &dep->deps; *pp != dep; pp = &(*pp)->next_ref)
            continue;
        *pp = dep->next_ref;
        if (tag_freed && dep->deps->refs == NULL)
            dep->deps->flags |= MOD_JUST_FREED;
    }
    /* And from the main module list. */
    spin_lock_irqsave(&module_list_lock, flags);
    if (mod == module_list) {
        module_list = mod->next;
    } else {
        struct module *p;
        for (p = module_list; p->next != mod; p = p->next)
            continue;
        p->next = mod->next;
    }
    spin_unlock_irqrestore(&module_list_lock, flags);
    /* And free the memory. */
    module_unmap(mod);
}

/* Called by the /proc file system to return a current list of
modules. */
int get_module_list(char *p)
{
    size_t left = PAGE_SIZE;
    struct module *mod;
    char tmpstr[64];
    struct module_ref *ref;
    for (mod = module_list; mod != &kernel_module; mod = mod->next) {
        long len;
        const char *q;
        #define safe_copy_str(str, len) \
        do { \
            if (left < len) \
                goto fini; \
            memcpy(p, str, len); p += len, left -= len; \
        } while (0)
        #define safe_copy_cstr(str) \
        len = strlen(mod->name); \
        safe_copy_str(mod->name, len); \
        if ((len = 20 - len) > 0) { \
            if (left < len) \
                goto fini; \
            memset(p, ' ', len); \
            p += len; \
            left -= len; \
        }
        len = sprintf(tmpstr, "%8lu", mod->size);
        safe_copy_str(tmpstr, len);
        if (mod->flags & MOD_RUNNING) {
            len = sprintf(tmpstr, "%4ld",
                (mod_member_present(mod, can_unload)
                 && mod->can_unload
                 ? -1L : (long)atomic_read(&mod->uc.usecount)));
            safe_copy_str(tmpstr, len);
        }
        if (mod->flags & MOD_DELETED)
            safe_copy_cstr(" (deleted)");
        else if (mod->flags & MOD_RUNNING)
            if (mod->flags & MOD_AUTOCLEAN)
                safe_copy_cstr(" (autoclean)");
            else if (! (mod->flags & MOD_USED_ONCE))
                safe_copy_cstr(" (unused)");
        else if (mod->flags & MOD_INITIALIZING)
            safe_copy_cstr(" (initializing)");
        else
            safe_copy_cstr(" (uninitialized)");
        if ((ref = mod->refs) != NULL) {
            safe_copy_cstr(" [");
            while (1) {

```

```

        q = ref->ref->name;
        len = strlen(q);
        safe_copy_str(q, len);
        if ((ref = ref->next_ref) != NULL)
            safe_copy_str(" ");
        else
            break;
    }
    safe_copy_str(" ");
}

static int s_show(struct seq_file *m, void *p)
{
    struct mod_sym *v = p;
    struct module_symbol *sym;
    if (!MOD_CAN_QUERY(v->mod))
        return 0;
    sym = &v->mod->syms[v->index];
    if (*v->mod->name)
        seq_printf(m, "%01x %s\t[%s]\n", (int)(2*sizeof(void*)),
            sym->value, sym->name, v->mod->name);
    else
        seq_printf(m, "%01x %s\n", (int)(2*sizeof(void*)),
            sym->value, sym->name);
    return 0;
}

struct seq_operations ksyms_op = {
    start: s_start,
    next: s_next,
    stop: s_stop,
    show: s_show
};

#else /* CONFIG_MODULES */
/* Dummy syscalls for people who don't want modules */
asmlinkage unsigned long
sys_create_module(const char *name_user, size_t size)
{
    return -ENOSYS;
}
asmlinkage long
sys_init_module(const char *name_user, struct module *mod_user)
{
    return -ENOSYS;
}
asmlinkage long
sys_delete_module(const char *name_user)
{
    return -ENOSYS;
}
asmlinkage long
sys_query_module(const char *name_user, int which, char *buf, size_t
    bufsize,
    size_t *ret)
{
    /* Let the program know about the new interface. Not that
       it'll do them much good. */
    if (which == 0)
        return 0;
    return -ENOSYS;
}
asmlinkage long
sys_get_kernel_syms(struct kernel_sym *table)
{
    return -ENOSYS;
}
int try_inc_mod_count(struct module *mod)
{
    return 1;
}
#endif /* CONFIG_MODULES */

        }
        while (!v->mod->nsyms);
        v->index = 0;
    }
    static void *s_next(struct seq_file *m, void *p, loff_t *pos)
    {
        struct mod_sym *v = p;
        (*pos)++;
        if (++v->index >= v->mod->nsyms) {
            do {
                v->mod = v->mod->next;
                if (!v->mod) {
                    unlock_kernel();
                    kfree(p);
                    return NULL;
                }
            } while (!v->mod->nsyms);
            v->index = 0;
        }
        return p;
    }
    return ERR_PTR(-ENOMEM);
}
unlock_kernel();
kfree(p);
return NULL;
}
static void *s_start(struct seq_file *m, loff_t *pos)
{
    struct mod_sym *p = kmalloc(sizeof(*p), GFP_KERNEL);
    struct module *v;
    loff_t n = *pos;
    if (!p)
        return ERR_PTR(-ENOMEM);
    lock_kernel();
    for (v = module_list, n = *pos; v; n = v->next, v = v->next) {
        if (n < v->nsyms) {
            p->mod = v;
            p->index = n;
            return p;
        }
    }
    return NULL;
}
}
unlock_kernel();
kfree(p);
return NULL;
}
static void *s_start(struct seq_file *m, loff_t *pos)
{
    struct mod_sym *v = p;
    (*pos)++;
    if (++v->index >= v->mod->nsyms) {
        do {
            v->mod = v->mod->next;
            if (!v->mod) {
                unlock_kernel();
                kfree(p);
                return NULL;
            }
        } while (!v->mod->nsyms);
        v->index = 0;
    }
    return p;
}

```